Supportive Methodology and Technology for Creating Interactive Art

by Greg Turner

M.Comp. (Hons) Loughborough University

A thesis submitted to:

Faculty of Information Technology

University of Technology, Sydney

Australia

For the degree of:

Doctor of Philosophy in Computing Science

Supervised by:

Professor Ernest Edmonds, UTS

Dr. Tim Mansfield, Industry

Submitted for Examination: December 20th, 2006

Final revision: May 18th, 2007

Statement of Sources

I, Greg Turner, declare that the work presented in this thesis is, to the best of my knowledge and belief, original, except as acknowledged in the text, and that the material has not been submitted, either in whole or in part, for a degree at this or any other university.

Signed:

Acknowledgements

This is the one part of the thesis I can be sure the examiners won't read, so I think it is safe to relax the rules of scholarship for two pages, because the justification for thanking these very special people need not be very rigorous. It's been a big journey, and my thanks are literally the size of the world: I shall begin with the far hemisphere and perform several circumnavigations.

Thanks heaps, as Aussies say, to mum and dad, for the selfless kindness and support, keenly felt from the other side of the planet, that only loving parents can give. I love you too! To Alankar, for reminding me that life is best when it's lived, and to Nick, for reminding me well that life isn't to be lived in books alone.

Formal, yet heartfelt, thanks to the institutions that have supported my study with their kind and forward-thinking scholarships, for what is unusual and difficult research. These institutions are the Faculty of IT, and the ICAN group at UTS.

Thank you to all the fine minds at CCS, but most of all to the desk quintet: Lizzie, who taught me the true meaning of true meaning and tidied up afterwards; Alastair, for setting the pace—in love as well as labour (both sorts!); Yun for blazing a trail with her strength and honesty; Deb for her ability to single-handedly arrange the universe such that everyone enjoys themselves (with Canada at the middle).

Like thanx or whatever to the juvey ledges that I like live with you know Claire, Amber and like Sebastian for looking after me loads and making life outside the matrix so like bodacious and like totally an in-joke haven. Woo, Collins St. rocks, you guys!

Gerhard, the most skilled, in-demand, non-programmer I know, gets a special nod for his support and his strange genius, and on behalf of the examiners, I thank him for making this document so easy on the eye.

Back on the other side of the world, here's to all of my real-life heroes who I haven't seen in too long: Ruth, who knows me better than anyone, and still thinks that's a good thing (likewise!). Robert and Lindsay, for sharing life's joys through great hospitality and the Socratic method; Russ and Jill for sharing life's joys through great hospitality and an excellent Bloody Mary. Jim, whose inspired and inspiring flex of phrase delights even more through its comparative rarity; Kate and Tim who are so big, and so much fun it was almost rude to get married on top of it all; and David, who more than anyone, embodies all the things I miss about Britain.

But for this thesis, of course, no-one deserves more thanks than Ernest Edmonds and Tim Mansfield, my supervisor and co-supervisor. Ernest, I will always admire you for your hard work, your taste, and for your willingness to make amazing things happen for people like me. Tim, you are friendlier, funnier, wiser and more quotable than any mentor I could have hoped for, and I love that the pace of learning hasn't slowed for you after your PhD.

And thank you finally to "the artists", particularly George, Dave, Norie, Maria, Petra, Keir and Daniel, without whom "all this" would be decidedly pointless, and much less fun.

Simple things should be simple.
Complex things should be possible

Alan Kay's law of simplicity.

…and there should be a smooth curve between them.

Greg Turner's corollary.

Abstract

Computation, as a medium for programming, supports scientists, mathematicians and "algorithmically-creative" (Amabile, 1996) workers very well. 'Deep' programming environments, with few, or flexible constraints, are designed for these kinds of computation. However, most artists, designers and other "heuristically-creative" (Amabile, 1996) workers must make do with more 'gentle' programming environments, such as Max/MSP or Processing, which support particular conceptual spaces well. Yet once the constraints of those spaces are come up against, they are found to be rigid.

The new media world is, by now, used to seeing interdisciplinary work that involves artists and technologists in collaboration, sometimes in response to this difficulty. These collaborations combine the power of artistic modalities of thinking with the full capabilities of computational media, but still the computing medium must be mediated for the artist by the technologist. Such mediation is at risk of reinforcing boundaries between artists and technologists, and denies artists 'hands-on' creativity in the medium, which is not only frustrating but also can destroy artistic meaning (Candy & Hori, 2003).

How can we make computational media better support creative workers, in and out of collaborations? My answer stems from the roles of constraints which surround conceptual spaces, but which can support creativity only as far as they can be changed in response to a change in conceptual spaces (Boden, 2004). Computation is an attractive medium because potentially supports highly changeable constraints. However, this potential is not realised—there are plenty of constraints within computing today which are neither inherent nor useful for creativity, but imposed as a result of industrial practices which are decreasingly relevant in today's techno-society. An example is the constraint around every compiled program preventing any modification of that program. Since these constraints cannot be changed in response to changing conceptual spaces, creativity is limited.

To remedy this technological disjunction between conceptual spaces and supportive media, I have made recommendations for future computing systems in which imposed constraints are not rigid. For example, if someone wishes to explore or change a particular constraint in such a computing system, they can 'lift the hood' and discover what's happening and change it, recursing if necessary to the level of computing fundamentals, but using a similar interface paradigm to that which they have already been using. Such a computing system allows people to change a computing medium to fit with their changing conceptual spaces.

To illuminate the accompanying social issues of supporting interdisciplinary collaboration, I carried out a grounded theory inquiry into the roles of collaborating experts—predominantly artist and programmer—working in interactive art collaborations. By studying first-hand reports and conducting interviews, I was able to build a rich theory of technology's role in the collaborative process. Most importantly, I found that non-programming artists prefer to use shared language and boundary objects (Fischer & Ostwald, 2003) that are also meaningful in computing terms. An example is when a programmer constructs 'computational toys', which sit between conceptual spaces and thus can be manipulated to create technical, aesthetic and computational meaning simultaneously.

To evaluate these findings, I synthesised the computing recommendations and the toy-making methodology, and examined prototypical examples of them in the light of a real-world art collaboration called *Cardiomorphologies v. 2*. The collaboration involved the development of several computational toys in the Max/MSP computing system, and also a technology for quickly creating toys.

Contents

1. Introduction

Chapter Overview

This chapter introduces my thesis, by defining its topic and scope, summarising my approach to tackling this topic, describing goals and describing how each chapter contributes to those goals.

The thesis statement is a very concise summary of my findings, the results of the new research in this thesis. This is followed by a concise statement of the approach I took to reach those findings, which is expanded upon in Chapter 3, the methodology. After this I introduce the motivation of my thesis at several levels: from supporting creativity, to

supporting art, to supporting interactive art, to the particular research question. The research question implies particular research goals (set within wider aims), which are described in the sections which follow. Finally, I present an overview of the outcomes of each chapter.

Thesis Statement

My thesis is that being able to alter constraints around technological and social zones of expertise is critical to effective creativity support in interactive art development and elsewhere. These constraints cannot be changed in conventional computing system designs, and are not supported in conventional methodologies. We can remedy this by:—

a) implementing a computing system in which technological constraints can be changed according to creative needs, because the system allows its own alteration, to a fundamental level

b) supporting this computing system through methodologies and technology in which constraining social boundaries can be changed according to creative needs, because they support, amongst other things, computational toy-making, a way to generate and use shared language which is meaningful not only across boundaries but also directly in computational terms.

Overview of My Approach

My approach has been principled, iterative and uses a variety of methods to illuminate this complex topic from several angles. I begin with a statement of the broad problem in this chapter, and progress to an examination about what we know about creativity support and about interactive art. After discussing the methodology, I reflect upon previous studies in creating interactive art. My expertise as a computer scientist has led me to critically examine existing technology in the first new study, to identify recommendations for areas for improvement. Throughout all of this work, the importance of social roles in interactive art collaborations became clearly the area in which significant research remained to be done, so for the second new study, I undertook an inquiry into these social roles, and produced a grounded theory, which describes them richly. I used this theory to specify technology and an

intertwined methodology for creating interactive art. Finally, I implemented these specifications, or prototypes of the more strategic ones, in a new interactive art collaboration, in order to test their validity.

Why Creativity?

Creativity support is a hot topic at the moment. Richard Florida observes in The Rise of the Creative Class (Florida, 2002) that our society and economy is becoming increasingly dominated by creative knowledge work, and that people are increasingly living their lives in the creative ways that scientists and artists always have. To put the argument quantitatively, the most current figures available at the UK Department for Culture, Media and Sport's site ("Creative Industries Economic Estimates Statistical Bulletin," 2005) show the 2003 value of the UK creative industries to be an 8% contribution to the economy (by my reckoning, £76.6bn), with an average 6% (11% for software) annual growth since 1997. Rising creativity means there is a new demand for tools, methods and environments that enhance, encourage or facilitate creativity.

And academia is responding to this demand. A 75-page report funded by the National Science Foundation of America, entitled Creativity Support Tools (Shneiderman, Fischer, Czerwinski, Myers, & Resnick, 2005), describes the consensus of researchers in the field, summarising established principles and important research issues. These researchers are psychologists, sociologists, artists and tool designers, all looking for ways to improve creativity support. The report's NSF-funded status means that western governments are seeing creativity support as a significant avenue for research. The challenges for future research listed in the report, reproduced in full here, include the following list. I have highlighted the issues which my research directly addresses in green:

- evolving existing and developing new theories of creativity (incorporating social, technical, and organisational dimensions) grounded in a deep understanding of creativity;
- identifying the fundamental role of creativity in all disciplines (science, design, engineering, art, business, education…);

- radically new creativity support tools that facilitate and enhance the development of creative thinking and creative expression grounded in the ongoing technology changes that will impact creativity;

- exploration and impact of these new creativity support tools in a broad spectrum of intellectual activities, including: problem framing and problem solving, decision making, collaboration, composition, visualisation;

- design of processes supporting creativity (based on what enhances or hinders creativity) including: the development of organised approaches to creativity that are grounded in the multi-dimensional character of creativity;

- the importance of end-user development for creativity; the impact of creativity on new divisions of labour;

- design of socio-technical environments to support and enhance creativity;

- systematic foundations for the design, assessment, and wide-spread distribution of creativity support tools;

- development of new assessment approaches (what should be measured and what can be measured) including: differentiation between quantifiable and qualitative dimensions;

- identification of qualitative dimensions such as: personally meaningful activities, mindsets, relevance;

- evaluation techniques applicable to ill-defined, open-ended problems;

- exploration and use of assessment and evaluation frameworks from different disciplines including: formal user studies, ethnographic studies analysis of social impact, cultural meaning;

- frameworks to educate the creative minds of the future by integrating knowledge about creativity into educational curriculum and professional training;

- new inter– and transdisciplinary collaborations focused on creativity including social and technological infrastructures to identify common ground and to create a shared understanding; understanding the role of diversity and distances (spatial, temporal, conceptual, technological) in creativity;
- studies of creative people and creative artefacts; creating repositories of creative artefacts to be studied and further evolved;
- understanding the importance of creativity in knowledge work, lifelong learning, and integration of working and learning.

One other point of consensus in that report is that if a researcher wants to create technology that supports creative people, he or she should study how creative people use tools. It makes sense for the researcher to study the kinds of people that he or she is designing for, and the tools that they currently use. In my case I have chosen to study artists using computers to make interactive art, for reasons I shall now explain.

Why Study Artists?

There is little reason to attempt to define artistic creativity here, because that would be the subject of several other theses, none of which would be universally agreed upon. My research is just as valid, and necessarily difficult, if we take the all-encompassing relativist view: if someone thinks that something or someone is creative, then that object or person is creative; if someone thinks an object or concept is art, then it is art. We should specify, in these circumstances, that art is creative. Further, that art-making is a quintessentially creative activity. So I study artists firstly because we are reasonably sure that they are creative, secondly because they are interesting to me, and thirdly because research involving art is productive—it has artistic outcomes, as well as technological or scientific.

(A small note on creativity and ethics: I adopt the view that, other things being equal, 'more' creativity is preferable to 'less'. I acknowledge that creativity can be used for good or bad purposes, but, like the assessment of what constitutes creativity itself, leave that to the discretion of the individual.)

Why Study Interactive Art?

For the purposes of this thesis, I define interactive art to be any art that responds to the presence of, or action by, participants in the artwork's audience. Each participant enters into a 'dialogue' with the system, to varying degrees of literalism and length. In this thesis, I will use the term to refer specifically to computer-enabled interactive art, as opposed to kinetic or electronic art (examples of which could elsewhere be considered to be interactive art). I make this specification because computer-enabled systems are capable of interaction in ways that can (with necessary augmentation) more-or-less subsume other mechanical or electronic forms, in addition to interacting in unique ways of their own, which shall be explored later (see Chapter 5).

Isn't all art interactive? Well, in the sense that the meaning of the art is made and changed by each person who perceives it through interacting with it, all art could be termed 'interactive'—park sculptures change role and meaning as children climb on them, for example. Yet interactive art, for my meaning of the phrase, changes actively (in ways that 'inert' art could not) and objectively (in ways that others can observe) in response to participants. Interactive art responds. As Christiane Paul puts it, "interactivity allows different forms of navigating, assembling or contributing to an artwork that go beyond [a] purely mental event" (Paul, 2002, p. 67). Interactivity is the "very medium of the work" (Muller, 2005). This makes it possible to show participants ideas, particularly ideas about themselves as individuals, which would be difficult or impossible to do otherwise.

Why did I choose to study "Interactive Art", rather than a category of art more generic, more well-known or simpler to define, such as "New Media Art"[1] , "'Net Art" (short for Internet Art), "Software Art" or "Digital Art"? Firstly my background as a computer scientist and interaction designer lends itself well to this particular topic. Secondly, as I

1 The phrase "new media art" describes art created with 'new' technology. At the time of writing, 'new' technology is that popularised after approximately the mid-20th Century, particularly computing and telecommunications technology. There are many debates, not directly relevant here, about the continuing role of new media in the arts, for example, about whether 'new' is still new, and whether new media is still a field distinct from other forms such as sculpture, dance, etc. A characteristic feature of new media art, in comparison with traditional forms of art, is its vulnerability to changes in the technology that it uses—curators are grappling with the problems of archiving new media art that requires specific non-permanent technology in order to work.

mention above and explain in the literature review, interactive art's unique properties can show us things about ourselves that we would not otherwise experience, which, empirically, appears to be an important factor in creativity support and, ideologically, I think is worth encouraging. Thirdly, interactive art is a stratum that runs through many of the other sub-disciplines within and without new media, and so, as a category, it resists cliché, pigeonholing and obsolescence as perhaps those sub-disciplines, or new media itself might not. Fourthly, and crucially, interactive art is one of the more complex manifestations of computer systems, involving not only a blend of art and Human–Computer Interaction (HCI), but also fairly advanced or esoteric exploitations of the technical capabilities of computers. The combination of innovative HCI with innovative computer science gives us exciting new types of encounter with the computer, which can influence the way we think about such encounters in the future[2]. The anthropologist Lucy Suchman elaborates on this theme (Suchman, 2002), contrasting it with attempts to replicated human behaviour in computers (which, she says, are doomed to failure since artefacts are too fundamentally different to humans). Basically, she points out that the aim should not be to design interactive artefacts that are personified—computing can be just as affective with interactive artistic artefacts. Affective computing (introduced by Rosalind Picard (Picard, 1997)) is computing which inspires an emotional response in users (potentially, says Picard, by 'feeling' emotions themselves).

The Research Question

My starting point is a quotation from the book Explorations in Art and Technology edited by Linda Candy and my supervisor, Ernest Edmonds:

> *A fundamental question that we have been considering is, what kind of environments best support the development of digital art? There is one answer to this question which, although it may sound a little strange, is, nevertheless, appropriate. In art and technology environments, we need environments for building environments. This*

2 Most successful interactive art can make this claim to some extent. A canonical example is the 'mystical' experiences of users of Char Davies' Osmose, explored in e.g. (Treadwell)

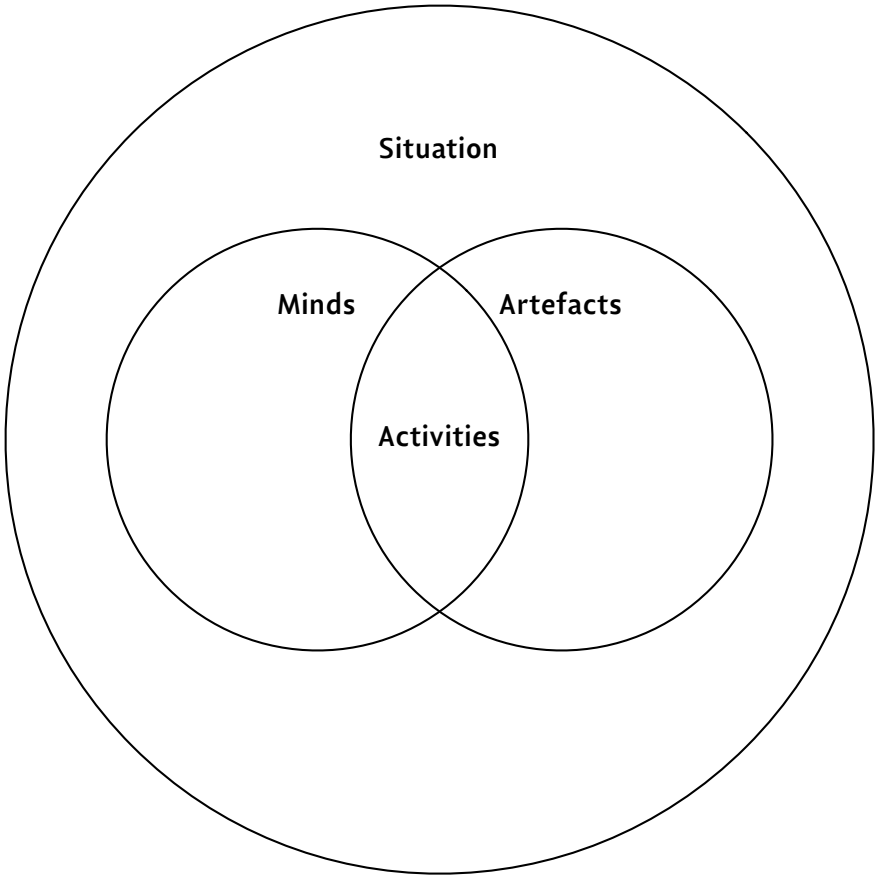
> *approach is analogous to having a store which stocks all of the components that one might need in order to build a carpenter's workbench. The store is an environment that has all of the components that one might need, such as vices, bench tops, tool racks, etc. By selecting from them and assembling the items in our own workroom, we can build a specific environment suitable for our particular carpentry needs. The store provides an environment for building the particular environments that its customers need.*
>
> (Candy & Edmonds, 2002)

Whilst this description of the problem and a potential solution may sound attractive, things are not necessarily so simple. In the first place, what is meant by an 'environment' in this context? In the second place, the flexible nature of computing and the innovative nature of digital art both mean that 'all of the components that one might need' is an effectively infinite and changing set.

CONSTRUCTING AN ENVIRONMENTAL FRAMEWORK

The question of what is an 'environment' perhaps is not so straightforward. In Candy's and Edmonds' book, an environment is the context in which creative activity takes place, and since we do not know precisely what creativity or context is, we cannot say precisely what an environment is and for whom. Instead we should ask what might constitute aspects of an environment for someone, constructing an individual's environment in terms of these aspects and their qualities. We can say subjectively, for instance, that an individual's creative environment might involve these aspects:- minds (maybe just the individual's mind)- artefacts- activities- everything else (which I shall call the situation)

Every mind, artefact and activity exists in a set of such situations (a concept which has echoes of Candy's and Edmonds' phrase 'environments for building environments'), each of which in turn has its own minds, artefacts and activities, and so on. In diagram form, an environment might look like this:



Minds refers to the personalities and thoughts of creative workers. I use the word *artefacts* to refer to the physical objects (including software) involved, and *activities* to refer to the interactions between minds and artefacts. The *situation* is admittedly a deferment of the question of 'what is an environment?', but we can say some useful things about it. The situation is all those intersecting environments—people, artefacts and activities and situations—that, in the view of an individual, pertain to the creative environment, but do not directly participate in it. Each quality of each of these aspects might be thought of as supporting creativity, inhibiting creativity or being neutral. My arrangement of the aspects of an environment gives a simplistic indication of some interactions between these aspects—for example, the situation affects everything (if the situation does not support creativity, this would have a negative impact on the ability of artefacts, activities and minds to support creativity, and vice versa); minds and activities affect each other; minds and artefacts may not affect each other except through activities, and so on. Conversely, each of the diagram's inner aspects may reflect

qualities of its containers. We can use this arrangement as a framework for recording and thinking about the different aspects of creative environments which we need to consider, and which will become useful to refer to as we progress.

The point is that rather than phrasing the problem or solution in a loosely-defined, subjective and confusing 'environment'-centric way, we can phrase them in terms of qualities of minds, artefacts and activities and let each individual construct their situation (and hence complete environment) to suit, which may be what they do anyway.

So how does this discussion of environments impact the research question? People who design creativity support technology should pay attention all aspects of a creative environment, but their design focus is on the artefacts aspect, and how those artefacts interact with the activities aspect (because every artefact has related activities). This means that it is not the goal of tools researchers to design minds or situations—that would be a task for environmental psychologists, if anyone. We can therefore choose, without having to design, the situation and people for whom we want to design creativity support artefacts and activities. My research question is an artefact- and activity-centric rewriting of Candy's and Edmonds' question "what kind of environments best support the development of digital art?", and focuses on interactive art instead of digital art. It is this:

Q. **In a situation which otherwise supports creativity, what would be useful technology and related methodology to help people to make interactive art?**

A situation that supports creativity is one that possesses all of the qualities that researchers have found to be useful for creativity support situations, which we shall come to in the Literature Review (Chapter 2). Useful technology comprises collections of artefacts that are likely to improve the process of creating interactive art by supporting creativity. Useful related methodologies are frameworks of activities that are likely to support the process of creating interactive art using that technology. People, in the context of my question, are those who want to make interactive art, and who possess the minds referred to in the creative environment.

Core Argument

In brief, my answer to the question stems from the roles of constraints on creativity—but it is important to note that constraints are not always opposed to creativity.

What is creativity? Briefly (a more detailed description is in the Literature Review, Chapter 2), creativity is the ability to produce work that is both novel and appropriate (Sternberg, 1999, p. 3), and such ability and work necessarily takes place within the liminal space within constraining boundaries around the zones of capabilities of minds and media. One might think at first that constraints always result in restrictions on creativity (I use 'restrict' in a pejorative sense), but creativity researchers have established that appropriately balanced constraints form stimulating spaces to explore. Too few constraints, too much freedom, can result in the "staring at the blank page" type of mental block, and too many constraints restrict creative activities to a frustrating degree. Margaret Boden (Boden, 2004, p. ch. 3, 1994, p. ch. 4) argues that creativity can always be ascribed to some generative system and so then to the constraints of that system, and concludes that "constraints—far from being opposed to creativity—make creativity possible. To throw away all constraints would be to destroy the capacity for creative thinking" (Boden, 1994, p. 79).

These constraints, together with the zones which they surround, are what Boden calls "conceptual spaces" (Boden, 1994), which she describes as "the organising principles that unify and give structure to a given domain of thinking" (Boden, 1994, p. 79). People, when they think, may explore these conceptual spaces in various ways (Boden gives the example of Johann Sebastian Bach's Well-Tempered Clavier as an exploration of the range of tone in the—then new—well-tempered tuning). As exploration takes place, the constraints of the space become apparent. Creativity, according to Boden, involves the transformation of these constraints, in simple ways through changing them, negating them, removing them or adding to them, and in more complex ways involving constraints from other spaces. For example, Arnold Schoenberg's dropping of the constraints of tonal hierarchies, used since the time of Bach, led to his development of the twelve-tone technique of musical composition. It follows that if constraints can be transformed

when they need to be, then they become a site for creativity. It also follows that if constraints cannot be transformed when they need to be, creativity is limited by these constraints.

This use of the terms 'constraints', 'zones' and 'conceptual spaces' as cognitive terms must remain somewhat metaphorical in this discussion, as we do not know precisely how they manifest in the brain; the approach that Boden and others have taken to rigorously define them has been to implement them as computational models, which does not currently well replicate human behaviour, and so is beyond the scope of this thesis. However, we can more readily identify particular constraints in the media we use.

A medium, plural media, is, in the broadest sense, any "extension of man" (McLuhan, 1964), and the constraints on a given medium are things that the technology, physics and economics of that medium permit, and things it does not. For example, the medium of acrylic painting allows acrylics to be painted with a brush, or with fingers, and so on. It does not allow, for instance, painting with oils instead (otherwise that would be oil-painting), or for the paint to float in mid-air (which would be impossible on Earth). Some media have relatively few, relaxed, constraints (for example, the medium of painting), and some media have relatively many, rigid, constraints (for example, the medium of painting children on their birthdays with shades of blue acrylic on a metre-wide canvas). Let us say that if a medium has all the constraints of another medium, plus more, then it is a sub-medium of that medium. So oil painting is a sub-medium of painting, and oil-on-canvas is a sub-medium of oil-painting. We could similarly say that Adobe Photoshop is a sub-medium of graphics software, and graphics software is a sub-medium of computing.

The constraints of media need to change as conceptual spaces change. Suthers writes: "We know that representational artefacts [often called by other names] are useful for offloading work, by serving as an external memory, and by translating cognitive operations to perceptual or mechanical operations. The act of expressing one's ideas in a representational notation helps one make them more precise and explicit: there is a dialectic that takes place between the person and the representation

during the process of representation." (Suthers, 2003, p. 300). If, then, a given medium becomes unable to represent ideas, it is no longer useful for "offloading work", or supporting those ideas, in other words.

The need for media to change in response to change in conceptual spaces is further complicated by situations in which several people are collaborating (a common scenario in interactive art development). As Fischer describes (Fischer, 2001; Fischer & Ostwald, 2003, pp. 220-224), people who collaborate on a particular problem belong to communities of interest, which means that there is a certain overlap to their individual conceptual spaces. Within these shared spaces, collaborators develop and use boundary objects, which are physical objects meaningful across individual spaces, to communicate with.

Computing is an attractive medium for art and other creative endeavours, because, as I discuss in chapter 5, it delivers the potential to support these highly-changeable constraints, partly because of its Turing Machine origins, partly because of its unique speed and servitude. However, this potential is not realised—there are plenty of constraints within computing today which are neither inherent nor useful for creativity, but imposed as a result of industrial practices, such as separation of concerns and speed of development, which are decreasingly relevant in today's techno-society. Since these constraints cannot be changed in response to changing conceptual spaces, creativity is limited. These imposed and rigid constraints result in a plethora of incompatible sub-media of computing, each sub-medium contained by imposed constraints. An example is the constraints around every compiled program preventing any modification of that program.

In other words, computing media cannot be changed nearly as much as they potentially could be to support the changing conceptual spaces of the people using them, which means that the constraints around conceptual spaces and computing media tend to become completely disjoint. Once a person's conceptual space diverges from a medium sufficiently, the medium is left behind, and the person must abandon his or her expertise in that medium, and acquire expertise in a new medium. For example, someone who wants to write a letter may find their needs served by any word processing program, but as their needs become

more and more elaborate and individual (particular colours, particular languages, particular shapes of paper), the variety of software that will support those needs diminishes. There will come a point where no existing compiled software will support the envisaged needs (there are, to my knowledge, no word processors that detect plagiarism in Morse code, for example). This has created a situation where making creative software is only possible or effective with the acquisition of expertise in multiple separate sub-medium. This expertise is currently gained in two ways—either by taking the time to learn the required knowledge (assuming the ability to learn that knowledge even exists), or by gathering a group of experts in each sub-medium who are collectively able to maintain sufficient shared zones of interest for communication to be successful.

To remedy this disjunction between conceptual spaces and supportive media, I have made recommendations for a future computing system in which imposed constraints are not rigid. For example, if someone wishes to explore or change a particular constraint in such a computing system, they can 'lift the hood' and discover what's happening and change it, recursing if necessary to the level of computing fundamentals, using a similar interface paradigm to that which they have already been using. This computing system allows people to change a computing medium to fit with their changing conceptual spaces.

To illuminate the accompanying issues of supporting constraint-changing human-human communication, I carried out a grounded theory inquiry into the roles of collaborating experts—predominantly in the roles of artist and programmer—working in today's interactive art collaborations. The inquiry is described in full in Chapter 6. By studying first-hand reports and conducting interviews with artists and programmers, I was able to build a rich theory of technology's role in the collaborative process. Most importantly, I found that non-programming artists prefer to use shared language and boundary objects that are also meaningful in computing terms. An example is when a programmer constructs 'computational toys', which sit between conceptual spaces and thus can be manipulated to create technical, aesthetic and computational meaning simultaneously. Computational toy-making is neglected in conventional methodologies, in part because current technology re-

quires a high time investment to construct such computational toys. This highlights a need for technology that makes the construction of computational toys much easier.

Goals

The research question implies one apparent goal, which is a set of specifications for technology and related methodology for making interactive art, based upon a combination of literature, observed interactive art practice and best design practice. But whilst it might be possible for me to use these methods to specify technology and methodologies for interactive art, that does not necessarily mean that I can state that such-and-such a technology or methodology is always "useful", or even that it has "improved the creative process". So as far as evaluation goes, the research question itself is idealised. We can reformulate the question in terms more appropriate to the evaluation of a design within the scope of a thesis.

Q: Do makers of interactive art prefer [such-and-such] technology and methodology to others that they are familiar with?

It is difficult to precisely tell the effect of the presence of a technology and methodology, since there is no way to observe a creative process then 'rewind' it, insert the new technology and methodology, press 'play' and compare the results. The difference must be subjective, at least at small and medium scales of deployment. By using the term 'prefer', I have chosen one factor that could be said to represent an improvement in the creative process—that the creative worker prefers it. Other questions I could have chosen are, for example, whether the tools or methodologies made the process 'quicker', 'cheaper', 'more prize-winning', and so on, but these are difficult to evaluate in the context of interactive art at small or medium scales.

To evaluate my studies, I synthesised the flexible-constraints computing specification and the toy-making methodology, to make design recommendations, and examined prototypical examples of designs cohering with those recommendations in the light of a real-world art collaboration called *Cardiomorphologies v. 2*. The collaboration involved the devel-

opment of several computational toys in the Max/MSP computing system, and also a technology for quickly creating toys. *Cardiomorphologies v. 2* succeeded from the collaborators' and audience's perspectives, and the artist felt that the computational toy-methodology was preferable to his previous way of collaborating.

In summary, my goals are:

1) to use best design practice and observed interactive art practice to make recommendations for tools and methodologies to support people who make interactive art, and

2) to show whether these tools and methodologies are useful by

a) drawing from the literature and

b) by asking people who use these tools and techniques if they preferred using them to other tools and techniques.

Wider Aims

These goals are steps towards some wider aims. If art is a quintessential form of creativity, and if computers are a flexible type of tool, and if you can please artists working with computers, you have a good chance of pleasing everyone who uses computers creatively. What do artists want to do with computers? Anything!

Alan Kay makes the point in an interesting way:

> *Just as [Marshall] McLuhan predicted, computers today are almost universally used—except by scientists—for just imitating paper. We're still doing the same thing the Summarians [sic] were doing in cuneiform in 3000 BC. The exceptions are those computer scientists and some physical scientists who use a computer for what it's really good at, which is making and understanding complex models of things.*
>
> (Frenkel, 1994, p. 14)

Whilst this comment might sound dated, in the light of the proliferation of computing used in today's temporal media, such multimedia comput-

ing is only available to programmers, or those with the resources to acquire software (which they cannot then modify themselves). Even with the extent of computer use in popular media, arguably the most important impact computing has had upon society has been in the sciences, where people have the knowledge and skills to use computing creatively.

Aiming towards creativity support using computers, in turn, has some ulterior motives. As well as the motives of 'art for art's sake', and of the contribution to the creative industries economy, there are educational and social advantages to creativity support. For instance, Ken Perlin, director of the New York University Media Research Laboratory argues for universal procedural literacy.

> *Citizens who can make things, he said, would understand that they can affect the world; if we institute programming languages as part of the general education of all, we will be helping to guide a revolution in the way people think and act.*
>
> (Pinckard, 2003)

Research Outcomes of Each Chapter

The outcomes of this thesis are the following:

Chapter 2: A literature review that describes the state of the art on creativity, and arranges creativity support qualities into the environmental diagram, introduced in this chapter. After this a review of interactive art practice aims to describe the technological qualities of interactive art. Finally a high-level review of research in end-user programming is presented, in order to situate the research in programming technology in my new studies and design recommendations (Chapters 5-9).

Chapter 3: A description of and justification for the various methods I bring to bear on the complex, interdisciplinary topic of supporting interactive art collaboration. The first part of Chapter 3 summarises my stance for this research as being contextualist. For the first study (Chapter 5), I will make an argument about the medium of computing based upon computing literature and programming history. For the second study (Chapters 6 and 7) I will conduct a grounded theory study into

artist–technologist collaborations. These studies will be synthesised into design recommendations for technology and methodology to support interactive art (Chapter 8). These design recommendations will be evaluated in a third study (Chapter 9), a new interactive art collaboration which involves these recommendations.

Chapter 4: A first-person account of my preliminary interactive art collaborations, *cubeLife* and *Séa.nce*, in order to give the reader an understanding of two interactive art collaborations, and to reflect upon the influence these collaborations had on my research.

Chapter 5: An argument in several parts. The first part is that computing is an appropriate medium for supporting creativity for many reasons, but primarily because of its potential to allow the changing of the constraints of its sub-media (programs, applications or tools). The second part is that we are presented with a different world, in which this is not the case, because the constraints of compiled software cannot be changed. I demonstrate, by using counterexamples, that the limitations of compiled software are not fundamental to the computing medium itself.

Chapter 6: The first part of the grounded theory study into the technologist's role in interactive art collaborations. In this part, I begin the open coding process of data from COSTART and COSTART 2, two projects which involved a series of intensive artist–technologist collaborations. The outcome of this chapter is a hierarchy of categories that relate to the research.

Chapter 7: The second part of the grounded theory study. I develop the hierarchy of categories into a set of interview questions, conduct the interviews, code the interviews and incorporate the codes into the hierarchy of categories. The categories that have a great deal of supporting data are saturated, and are included in the theory; others are discarded. This chapter contains a write-up of the grounded theory.

Chapter 8: Together, the new understandings provided by Chapters 5-7 provide a holistic insight into the problem of designing computing systems and creativity support tools for interactive art. I use these new understandings to make recommendations for a single universal computing

medium which supports creative engagement, methodologies for collaborating over such a computing medium, and tools to aid these methodologies.

Chapter 9: An account of a new artwork collaboration which involves prototypical methodology and technology which implements the recommendations in Chapter 8, and an evaluation of that collaboration.

Chapter 10: A summary of the findings of the thesis, an answer to the research question, and suggestions for further work.

Summary

I introduced a Venn diagram of a notional creativity support environment in order to structure a research question using it. My research question is:

Q: **In a situation which otherwise supports creativity, what would be useful technology and related methodology to help people to make interactive art?**

I will approach this question from various angles. Firstly, I need to understand what is meant by "creativity support" and by "interactive art", from technological and methodological perspectives. Secondly I will use this understanding to recommend supportive technology and methodology. Thirdly I will evaluate my recommendations by implementing this support in a new interactive art project.

2. Literature Review

Introduction

Having introduced the topic of research in the last chapter, I will flesh out its four main themes in this chapter. Firstly, I will review the topic of creativity, since without doing so the concept is prone to being treated vaguely. Secondly I will review creativity support qualities, which are qualities of situations, minds, artefacts and activities that re-

searchers say are supportive of creativity. These qualities will be arranged on the Venn diagram introduced in Chapter 1, and are used to analyse technology and methodology (Chapters 5-7) and heuristically evaluate design recommendations (Chapters 8 and 10).

Thirdly I will review the history of interactive art, and the styles and technologies used in interactive art today. This is to give a sense of its broad variety and high demands on technology. I will use this in Chapter 5 to argue for the inadequacy of existing programming environments to support interactive art development.

Finally, in anticipation of using it in discussing technological support for programming interactive art (Chapters 5, 7 and 8), I will review general principles and styles of current End-User Programming (EUP) research.

Creativity

A good place to start in the search for knowledge about creativity is with the various authors of Robert J. Sternberg's *Handbook of Creativity* (Sternberg, 1999, p. 3), which tells us that creativity is the ability to produce work that is both novel and appropriate (though as Lubart points out (Lubart, 1990), this is a predominantly western view—eastern cultures tend to view creativity as more of a self-growth process).

The book's authors also explain at some length that creativity is typically described in terms of its results, because we do not know enough about the mechanism of creativity to describe it more directly. What's more, the criteria for work, novelty and appropriateness vary from individual to individual, and from researcher to researcher. In the concluding chapter, Richard E. Mayer compares the different meanings of novelty and appropriateness (actually using the terms 'originality' and 'usefulness', which are less appropriate to this research because of the implications that originality must be judged by society and that art has to be useful), and proposes some clarifying questions, so that researchers can say more clearly what type of creativity they are researching (see p. 100 for my answers to these questions):

1. Locus of creativity—is creativity a property of people, artefacts, and/or processes? Broadly speaking, authors focus on either the

differences between people (approaches, personalities) or artefacts (case studies or computer simulations), or on the steps involved in creative thinking or teaching creative thinking. As many researchers are realising (e.g. Wehner, Cszentmihalyi and Magyari-Beck (1991)—see just below), a more holistic approach to creativity is becoming necessary.

2. Relevance of creativity—is creativity a personal, social, societal or cultural phenomenon? Gardner's (1993) distinction between little C and big C creativity to indicate personally-relevant and culturally-relevant creativity lies along the same lines as Margaret Boden's P-creativity and H-creativity (P and H being short for psychological and historical creativity) (Boden, 2004), in that both views recognise that what may be creative in one context may not be creative in a wider context. Csikszentmihalyi (1999a; 1999b) characterises creativity as an inherently social system, that "creativity is as much the result of changing standards and new criteria of assessment as it is of novel individual achievements" (1999b, p. 321).

3. Frequency of creativity—is creativity a common or rare event? This question is tied to the question of whether creativity is part of normative cognition, or whether normative cognition does not include the type of creativity we see in eminent individuals such as Michelangelo or Einstein. Treating creativity as a rare event makes it more meaningful to search for common personality traits and for prerequisites for creative activity, but it is harder to find examples. Treating it as a common event makes it easier to gather data, but harder to find clear commonalities.

4. Specificity of creativity—is creativity domain-general or domain-specific? Mayer writes, "according to the domain-general view, creativity is a general skill or trait or characteristic that can be applied to a wide variety of situations. … [T]he domain-specific view of creativity is that different kinds of creativity are required in different domains … [T]he dominant view among cognitive scientists (Gardner 1983. 1994) is that cognitive ability is at least partially domain-specific" (Mayer, 1999, p. 451).

5. Measurement of creativity—is creativity qualitative or quantitative? The qualitative view is that creativity is unique to the individual, and can thus be understood only subjectively. The more dominant quantitative view is that "creativity consists of one or more factors of which people may have varying amounts" (Mayer, 1999, p. 451) and is measured psychometrically.

The variety of possible answers to these questions shows that creativity is viewed differently in different fields. Wehner, Csikszentmihalyi and Magyari-Beck (1991) examined a hundred dissertations on creativity in various fields. A significant duality that they exposed was that psychological texts treated creativity in terms of innovation, and business texts treated innovation (as the topic) in terms of organisation. Other disciplines treated creativity in their own ways—historically, sociologically, pedagogically, and so on. They compared these unidisciplinary approaches with the fable of the blind men and the elephant—none could appreciate the subject in its entirety. Hence they recommend that creativity be studied in a multidisciplinary way, with attention paid to many aspects of creativity.

The field of creativity support embraces this multidisciplinary approach. Psychologists describe the creative process, sociologists situate creativity in a social or historical context, business analysts situate creativity in an organisational context, artists and designers are finding creative ways to support creativity, and so on. There appears to be a good deal of interdisciplinary discourse between creativity support workers, which I aim to exploit in this research. There are many models of creativity and design, and I shall describe some of the most pertinent below.

Margaret Boden, a cognitive and computing scientist, draws a careful distinction between new ideas that are creative, and new ideas that are "merely new" (Boden, 2004, p. 1). This distinction is based upon an argument that ideas are the product of some generative system; the question of whether an idea is *creative* is the same question as whether the generative system that produced that idea is new. As she puts it:

> *We can now distinguish first-time novelty from radical originality. A merely novel idea is one that can be described and/or produced by the*

> *same set of generative rules as other, familiar ideas. A radically original, or creative, idea, is one which cannot*
>
> (Boden, 2004, p. 51).

Boden goes on to describe ways in which radically original ideas can be generated, which centre on the notion of *conceptual spaces* (e.g. Boden, 1994), which she describes as "the organising principles that unify and give structure to a given domain of thinking" (Boden, 1994, p. 79).

Conceptual spaces can be explored or transformed in order to produce radically original ideas. Boden gives the example of Johann Sebastian Bach's Well-Tempered Clavier as a creative exploration of the conceptual space of the—then new—well-tempered tuning. As exploration takes place, the constraints of the space become apparent. The generation of radically original ideas, according to Boden, involves the transformation of these constraints, in simple ways through changing them, negating them, removing them or adding to them, and in more complex ways involving constraints from other spaces. For example, Arnold Schoenberg's dropping of the constraints of tonal hierarchies, used in mainstream classical music since the time of Bach, led to his development of the twelve-tone technique of musical composition. It follows that if constraints can be transformed when needed, those constraints support creativity at those boundaries. It also follows that if constraints cannot be transformed when needed, creativity is limited by these boundaries.

Theresa Amabile (1983) describes creativity being maximised at the confluence of intrinsic motivation, domain-relevant knowledge and abilities, and creativity-relevant skills (Figure 1).

Gruber (1989) proposes a model of creativity based on the concept of evolving systems. A person's knowledge, purpose and affect develop as that person ages, and differences between individuals become amplified as time passes. This development can be observed in the work of eminent creative people, and is consistent with the challenging view given by Robert W. Weisberg (Robert W. Weisberg, 1999). He writes:

> *The relation between creativity and knowledge is much more straightforward than theories of creativity typically assume. One may*

> *be able to understand creative thinking by determining the knowledge that the creative thinker brings to the situation he or she is facing. The reason that one person produced some innovation, while another person did not, may be due to nothing more than the fact that the former know something that the latter did not*
>
> (Robert W. Weisberg, 1999, p. 248).

Weisberg's implication is that creativity is nothing more than thinking, and that we should strive for a theory of thinking. This, of course, broadens the scope of creativity support.

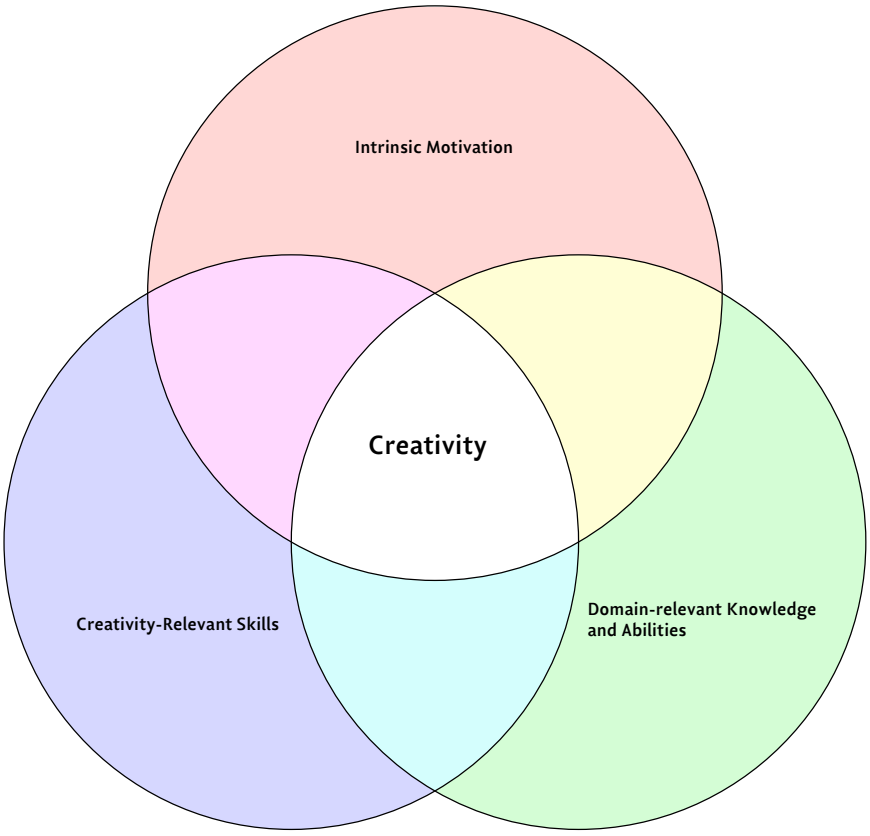


Figure 1. Theresa Amabile describes creativity as arising at the confluence of three areas: intrinsic motivation, creativity-relevant skills and domain-relevant knowledge and abilities.
(Amabile, 1983)

Csikszentmihalyi (Csikszentmihalyi, 1999b; and in R. A. Wilson & Keil, 2001), on the other hand, says that creativity occurs when all of the processes in his systems model of creativity take place. The system, comprised of the domain ("which consists of a set of rules and practices") the individual ("who makes a novel variation in the contents of the domain") and the field ("which consists of experts who at as gatekeep-

ers to the domain and decide which novel variation is worth adding to it"), represents a much more socially-situated view of creativity (Figure 2). Csikszentmihalyi says that "in the sciences as well as in the arts, creativity is as much the result of changing standards and new criteria of assessment as it is of novel individual achievements" (Csikszentmihalyi, 1999b, p. 321).

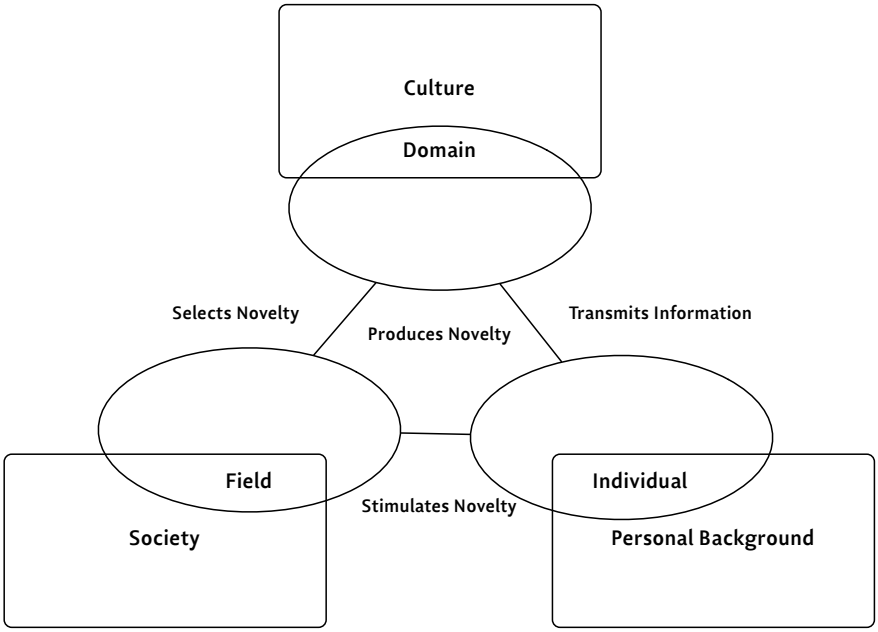


Figure 2. Mihalyi Csikszentmihalyi argues that creativity depends on a system in which individuals, society and culture are all key parts.
(Csikszentmihalyi, 1999b)

At first glance, Csikszentmihalyi seems to be referring here to H-Creativity or big C creativity rather than the personal level; however there is an implicit assumption that the range of the field will vary from situation to situation—for example the field of experts in how to stop one's daughter from crying is of different size and stature to the field of experts on quantum mechanics. Additionally the domain may be more short-lived than others. To take a concrete example, the goal might be to stop one's daughter from crying now, rather than to embark on a detailed scientific study, so novel variations may not enter into posterity. Given awareness of these variations, Czikszentmihalyi's description is applicable to all spheres of creativity.

The artist Harold Cohen (whose software was used by Boden as an example (Boden, 2004)) in *Colouring Without Seeing: a Problem in Machine*

Creativity studies the qualities of what he terms "Behaviour X" in order to avoid using the apparently ill-defined term 'creativity', using the charming example of his daughter's development of this behaviour, but with a focus on machine creativity. He concludes there are three main elements:

> *First, emergence, which implies a technology complex enough to guarantee it. […] The second element is awareness, […] but not necessarily conscious awareness, of what has emerged [Cohen uses this to differentiate naïve and mature activities—perhaps the prototypical analogue to P- and H-creativity]. Third—without which, in fact, we could hardly know that awareness had occurred—a willingness to act upon the implications of what has emerged; and this surely implies a distinctive psychology and motivation in the individual*
>
> (Cohen, 1999).

Sternberg and Lubart (1995) characterise creativity as a kind of investment in ideas, the theory being that if one 'buys low', and works on unpopular or esoteric ideas, developing them, one can later 'sell high', capitalising on the creativity. Sternberg and Lubart identify six resources which each need to be tapped in order to build valuable ideas:

1) intellectual abilities, comprising the synthetic ability to see problems in new ways, the analytic ability to recognise which ideas are worth pursuing, and the ability to persuade others of the value of one's ideas,

2) knowledge required to move beyond a field,

3) different styles of thinking global and local,

4) characteristic personality traits (see p. 32),

5) intrinsic motivation (see p. 31) and

6) a supportive environment in which to be creative.

An heuristic model of creative cognition is the 'Geneplore' model proposed by Ward, Smith & Finke (1999). Geneplore is a portmanteau of 'generate' and 'explore', being the two stages of this model. The genera-

tive stage involves retrieval from memory, formation of associations or combinations, synthesis of new structures, transformation of existing structures into new forms, analogical transfer of information from one domain to another and categorical reduction. The exploratory stage involves searching for novel or desired attributes, metaphorical implications, potential functions, evaluation from different perspectives or within different contexts, interpretation as solutions and the search for limitations suggested by the structures.

According to Czikszentmihalyi's definition in particular, and implicit in others, creativity is subjective to members of the field, which can only be objectively measured by monitoring social consensus or through psychometric testing. Social consensus ("67% think that this person is creative") is weakened by the fact that social opinion changes over time (van Gogh selling only one painting in his lifetime is a canonical example of this). Psychometric testing is normally only valid for one 'type' of creativity, as described by the test topics, and comparing the differences between different types of creativity comes down once again to a qualitative assessment (Hewett et al., 2005). In the next section, I have gathered together qualities which researchers in creativity support say encourage or enhance creativity. I have examined both creativity in general, and creativity in art and design practice in particular.

Creativity Support Qualities

I have already said in the introduction that we are not too concerned with 'designing' situations and people, as, for the purposes of my research, we can treat those as given. Yet I have also argued that, in order to design artefacts which support creativity, it is still important to understand as much as possible about the rest of the environment. As we proceed we can place the qualities described by the literature into the framework diagram introduced earlier, and mark them with underlined text. The completed diagram appears at the end of the chapter (p. 90)

CREATIVITY SUPPORT QUALITIES OF ETHNIC AND INSTITUTIONAL CULTURE

Different cultures place different values on creativity (Lubart, 1990; Maduro, 1976; Potter, 1996; Silver, 1981). For example, western cultures tend to place more emphasis on external aspects, such as achieving

breakthroughs in technology and methods, whereas eastern cultures tend to place more emphasis on internal aspects, such as self-growth. Japanese creativity, for example, has a strong focus on paradigm-preserving, through incremental refinement of paradigms (Proctor, Tan, & Fuse, 2004). This work is directed at supporting western-style creativity, though there is no particular reason to assume that the findings will be of no use in other cultures.

As we will shortly see, extrinsic motivation is an important enabling or disabling factor in creativity. People who are creative on a professional basis will normally have their extrinsic motivation provided in part by the organisation of the institution of which they are a part. As Wendy Williams and Lana Yang write (1999), there are several factors in conventional organisations that inhibit creativity. Conventional organisational structure stems from Smith's 1776 concept of division of labour (where tasks are split amongst people working in specialities) and through Weber's refinement, published in 1922, grouping labour into jurisdictional areas and hierarchies of supervision, traditional organisations encourage "rigid adherence to rules and regulations. This emphasis encourages conservative thinking and can hinder effective problem solving and information flow" (1999, p. 375). Innovation is overshadowed by existing tasks and rank.

Williams and Yang also criticise the machinery of bureaucracy, including the personality of individuals ("Individuals with a bureaucratic personality are concerned mostly with attaining the security of tenure and salary … concepts such as innovation threaten the stability that ensures these individuals' future within the organisation" (Williams & Yang, 1999, p. 376)) and rapidly-changing technology ("It may be difficult for workers to develop and advance creative ideas when most of their time and mental energy is spent simply becoming familiar with the basics of their jobs" (p. 375)). They suggest ways that organisations might change to better support creativity:

> *organisational environments fostering creativity share the following characteristics … : considerable freedom, good project management, sufficient resources, encouragement, an atmosphere of cooperation and collaboration, ample recognition, sufficient time for creative thinking, a*

> *sense of challenge, and internally generated pressure to accomplish important goals*
>
> (Williams & Yang, 1999, p. 383).

For completeness, it is worth mentioning Ted Levitt's article (2002/1963) *Creativity is Not Enough* in which he points out that (in business) creative ideas are worthless without taking into account practical matters of implementation. In that article he discusses a cerebral form of creativity—having ideas and thinking—and laments that ideas without implementation are harmful to business. In contrast, the definitions of artists and scientists described below are based upon product. However, Levitt has a point in drawing out the need for management of creative people, who may require some organisational support if they are unable to manage it themselves.

Raymond S. Nickerson admits that "a clear, unequivocal and incontestable answer to the question of how creativity can be enhanced is not to be found in the psychological literature" (Nickerson, 1999, p. 407), though he makes suggestions based upon agreement between researchers. Nickerson's suggestions

- Establishing Purpose and intention: find ways of getting people to intend to be creative
- Building Basic [Creativity] Skills
- Encouraging Acquisition of Domain-Specific Knowledge
- Stimulating and Rewarding Curiosity and Exploration
- Building Motivation (Especially Intrinsic)Encouraging confidence and a willingness to take risks
- Focussing on Mastery and Self-Competition
- Promoting Supportable Beliefs about Creativity (i.e. the above)Providing Opportunities for Choice and Discovery
- Developing Self-management (metacognitive) skills

- Teaching techniques and strategies for facilitating creative performance
- Providing Balance (not too much of anything)

CREATIVITY SUPPORT QUALITIES OF MOTIVATION

Motivation is an important factor in creativity support—an individual must want to be creative in order to be creative. Teresa Amabile is a prolific researcher into the psychology of motivations for creativity. She draws the distinction between intrinsic and extrinsic motivations (e.g. Amabile, 1987), and shows that the relationship between them is complicated.

Intrinsic motivation is characterised by Amabile as doing 'something you love', an intense interest in and curiosity about a particular topic. Torrance (1987) found that children doing something they loved were more creative. Csikszentmihalyi (1999a) describes this state as 'flow', where the challenges match a creative person's level of skill.

Intrinsic motivation is necessary for creativity, and extrinsic motivation has a relationship with it, sometimes synergistic, sometimes undermining. People who are motivated by extrinsic things such as expectation of evaluation or reward, task constraint, surveillance and competition result in less creative output (as judged by a panel of artists) than those who do not have these motivators. On the other hand, extrinsic factors such as non-contingent reward, recognition and feedback that confirms competence improved creative output.

To illustrate the complexity of extrinsic motivation, let's look at the role of reward. In one study, children were rewarded with the opportunity to play with a Polaroid camera prior to telling a story.

> *When children made a deal with the experimenter to tell a story in return for playing with the camera, the typical undermining effect of a reward was found [that the stories were less creative]. Surprisingly, however, children who told a story after playing the camera as a non-contracted-for reward actually told more creative stories than a control group.*
>
> (Collins & Amabile, 1999, p. 304)

Amabile (1983) suggests that extrinsic motives could divide people's attention between their extrinsic goals and the task at hand. Amabile suggests some practical implications for the workplace and classroom. The first of these is the freedom to choose what to work on. The second is to increase the importance of intrinsic motivation, by discussing it, acting on it and promoting it. The third is to decrease the importance of extrinsic motivations, by reducing their salience or by changing their character. Amabile suggests that extrinsic motivations will do less harm when the work reaches a less creative stage.

CREATIVITY SUPPORT QUALITIES OF HUMAN MINDS

Psychologists have found certain personality traits that characterise creative people. Based on a review of the literature, Barron & Harrington (1981) included in this list the traits of independence of judgement, self-confidence, attraction to complexity, aesthetic orientation and risk-taking. Lubart (1990; Lubart & Sternberg, 1995) identifies creative personality traits of self-efficacy, willingness to overcome obstacles [which is similar to motivation], to take sensible risks and to tolerate ambiguity.

	Nonsocial traits				Social Traits	
	Openness to experience, fantasy-oriented, imagination	Impulsivity, lack of conscientiousness	Anxiety, affective illness, emotional sensitivity	Drive, ambition	Norm doubting, nonconformity, independence	Hostility, aloofness, unfriendliness, lack of warmth
Alter (1989)	✔			✔		
Amos (1978)					✔	
Andreasen & Glick (1988)			✔			
Bachtold & Werner (1973)	✔	✔			✔	
Bakker (1988)				✔		
Bakker (1991)		✔	✔	✔		
Barron (1972)	✔	✔	✔		✔	
Barton & Cattell (1972)	✔	✔			✔	✔
Cross et al. (1967)	✔	✔	✔	✔	✔	✔
Csikszentmihalyi & Getzels (1973)	✔		✔	✔	✔	
Domino (1974)	✔			✔	✔	
Drevdahl & Cattell (1958)		✔	✔	✔	✔	✔
Dudek (1968)			✔			
Dudek et al. (1991)		✔		✔	✔	✔
Eiduson (1958)	✔		✔	✔		
Eysenck (1995)						✔
Feist (1989)	✔					
Getzels & Csikszentmihalyi (1976)	✔	✔	✔	✔	✔	✔
Götz & Götz (1979)		✔	✔			✔
Hall & MacKinnon (1969)	✔	✔	✔		✔	✔
Hammer (1966)			✔	✔		
Hammond & Edelmann (1991)		✔	✔			✔
Helson (1977)		✔	✔	✔	✔	
Holland & Baird (1968)	✔				✔	
Jamison (1993)			✔			
Kemp (1981)	✔		✔	✔	✔	
Ludwig (1995)			✔			
MacKinnon (1962)	✔				✔	
Marchant-Haycox & Wilson (1992)			✔	✔		✔
Martindale (1975)	✔		✔			
Mohan & Tiwana (1986)			✔			
Mohan & Tiwana (1987)		✔				✔
Pufal-Struzik (1992)	✔	✔			✔	
Richards (1994)						
Richards & Kinney (1990)			✔			
Rossman & Horn (1972)	✔				✔	
Schaefer (1969, 1973)	✔	✔	✔	✔	✔	✔
Shelton & Harris (1979)	✔		✔		✔	
Walker et al. (1995)	✔	✔	✔			
Wills (1983)			✔			
Wilson (1984)			✔	✔		✔
Zeldow (1973)	✔	✔			✔	

Table 1. Summary Feist's review of the literature of personality differences between artists and non-artists
(Feist, 1999)

Feist (1999) surveyed the literature (including Barron's & Harrington's work), defining artists to be (in my paraphrasing) those with aesthetically-driven creative output, such as photographers, painters, writers and so on. Feist surveyed literature comparing artists with non-artists, and comparing creative with less creative scientists. I have redesigned his linear tables of results into a matrix (Tables 1 and 2).

	Nonsocial traits		Social Traits	
	Openness to experience, flexibility of thought	Drive, ambition, achievement	Dominance, arrogance, hostility, self-confidence	Autonomy, introversion, independence
Albert & Runco (1987)		✔		✔
Bachtold & Werner (1972)			✔	✔
Bloom (1956)		✔		✔
Busse & Mansfield (1984)				✔
Chambers (1964)		✔	✔	✔
Davids (1964)		✔	✔	✔
Erickson et al. (1970)		✔	✔	✔
Feist (1993)		✔	✔	
Feist & Barron (1996)	✔			
Gantz et al. (1972)			✔	
Gantz, Erickson & Stephenson (1972)		✔		
Garwood (1964)	✔		✔	✔
Gough (1961)	✔	✔	✔	
Ham & Shaughnessy (1992)			✔	
Helmreich et al. (1980)		✔		
Helmreich et al. (1988)		✔	✔	
Helson (1971)	✔			✔
Helson & Crutchfield (1970)	✔		✔	✔
Holland (1960)		✔		✔
Ikpaahindi (1987)		✔		
Lacey & Erickson (1974)		✔	✔	✔
Parloff & Datta (1965)	✔		✔	✔
Parloff, Datta, Kleman & Handlon (1968)	✔		✔	
Roco (1993)	✔			✔
Roe (1952)				✔
Rossman & Horn (1972)	✔		✔	✔
Rushton et al. (1983)		✔	✔	✔
Schaefer (1969)	✔	✔	✔	✔
Shapiro (1968)	✔	✔	✔	
Simon (1974)		✔		
Smithers & Batcock (1980)				✔
Terman (1955)				✔
Van Zelst & Kerr (1954)	✔	✔	✔	✔
Wispe (1963)	✔	✔	✔	

Table 2. Summary of Feist's review of the literature of personality differences between creative and less creative scientists (Feist, 1999)

Both artists and scientists have relatively high introversion and independence, which in extremes can manifest as hostility or arrogance, but is also connected to the widely observed need for isolation or withdrawal, and for drive and self-confidence in creative achievement. In particular, Barron (1988) suggests that creative people distance themselves from others to avoid the negative extrinsic factors, such as surveillance or evaluation, making it important to remove those factors in collaborative situations.

Feist remarks that artists appear to be more anxious, emotionally adaptable and impulsive than non-artists and scientists. Barron & Harrington (F. Barron & Harrington, 1981, p. 456) remark that creative scientists are

more emotionally stable, venturesome and self-assured than the average individual. These things indicate a "disposition toward intense affective experience" (F. Barron & Harrington, 1981, p. 283) in artists. Feist comments,

> *This is not to say that creative process in art is exclusively emotional and in science exclusively nonemotional … The discovery stages of scientific creativity are often very intuitive and emotional, just as the elaboration stages of artistic creativity can be very technical and tedious*
>
> (Feist, 1999, p. 283).

Artists and scientists tend to be nonconformist, artists more so than scientists, as indicated by their low socialisation and low conscientiousness personality scores. Scientists are more organised and feel more responsible than artists, which, says Feist, is consistent with the notion that scientists are less actively nonconforming than artists. According to Sternberg and Lubart's investment theory of creativity (Sternberg & Lubart, 1995) mentioned earlier, a willingness to defy the crowd is required to be able to 'buy low and sell high'. This nonconformity could be related to the creative need to break constraining boundaries around conceptual spaces highlighted elsewhere in the literature (c.f. Candy & Edmonds, 2002; Boden, 2004).

There is a higher incidence of synaesthesia in artists, poets and novelists than in the population at large (Ramachandran & Hubbard, 2003; Ramachandran & Hubbard, 2001a, 2001b). Synaesthesia is a neurological condition in which a person experiences sensations in one 'sense', when another 'sense' is stimulated. For example, a synaesthete may 'see' colours in response to musical sounds. Ramachandran and Hubbard postulate, based on experimental evidence, that synaesthesia involves a 'cross-wiring' of the regions of the brain that deal with these different senses, and that some brains are more generally cross-wired than others. They use this to explain the high incidence of synaesthesia in creative people; in short, creative people have a remarkable ability to construct links between two seemingly unrelated realms, which could be assisted by a cross-wired brain. Less-synaesthetic brains have the ability to interpret these links:

> *When Shakespeare writes 'It is the East and Juliet is the sun', our brains instantly understand this. You don't say, 'Juliet is the sun. Does that mean she is a glowing ball of fire?' (Schizophrenics might say this; they often interpret metaphors literally). Instead, your brain instantly forms the right links, 'She is warm like the sun, nurturing like the sun, radiant like the sun' and so on.*
>
> (Ramachandran & Hubbard, 2001b, p. 17)

So it seems that creative output is related to personality traits, and that some types of people are more creative (judged by quantity of output) than others.

A small diversion: It is also possible to observe different physiological responses to different types of creative work. In particular, heart rate increases gradually with convergent-thinking problems, yet leaps suddenly with insight in divergent-thinking problems (Jausovec & Bakracevic, 1995). Although such responses are only peripherally relevant to design for creativity, it is heart-warming [pun intended] to note that two of the artworks discussed in this thesis use heart rate as an input device.

CREATIVITY SUPPORT QUALITIES OF ACTIVITIES

In her groundbreaking ethnography of computing systems design at Xerox PARC (Suchman, 1987), Lucy Suchman made the important distinction between *plans* and *situated actions*. Our behaviour is dictated by situated actions—action made in response to a situation—much more than by plans (besides, plans are a form of situated action). To use her colleague's example, a plan is similar to observing a river from a cliff top, before canoeing down it. In actuality, the behaviour of the canoeists is contingent upon the changing, unpredictable river *whilst* it is being canoed, and the behaviour of the other canoeists, and so on. It follows that creative activity is generally unplanned and unpredictable, so designed artefacts need to minimise their dependence on a planned activity.

In Amabile's work (Amabile, 1983), creative activity was characterised as the confluence of intrinsic motivation, domain-relevant knowledge and abilities and creativity-relevant skills. These skills are coping with complexities, breaking one's mental set during problem-solving, heuristics

for generating novel ideas, concentrated effort, high energy, the ability to set aside problems.

Creativity can be characterised as the manipulation of problems. Csikszentmihalyi, for example, says that creativity is more about problem finding than problem-solving (Csikszentmihalyi, 1999b). In other words, a problem that requires a creative solution is often not very well defined, and one outcome of the creative process may be a better-defined problem. To take a concrete example, the problem of "how can I make my painting convey the changing impression I have of this landscape?" may be refined by an impressionist painter to become "visible brushstrokes convey changing light. What colours should I choose for this light?" and so on.

The role of insight in response to these ill-defined creative problems is similarly elusive. Weisberg and Alba (1981) and Baker-Sennett and Ceci (1996) claim insight is essential for creative work. The test problem Weisberg and Alba used in their experiment was how to connect 9 dots arranged in a square grid with exactly four straight lines joined end to end (Figure 3. The solution is at the end of the chapter).

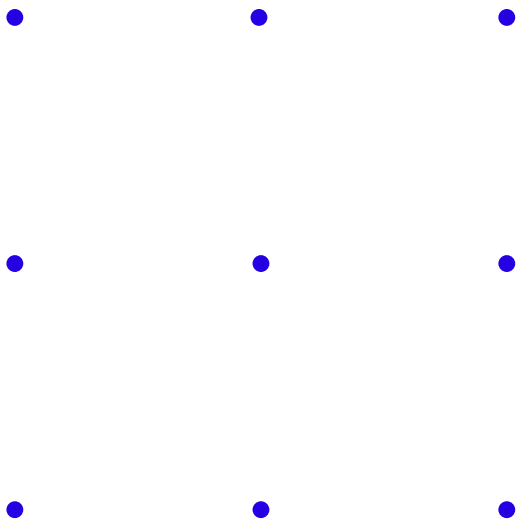


Figure 3. The 9-dot problem which Weisberg and Alba (1981) used to test for insight. The problem is to connect all 9 dots with exactly four straight lines joined end to end.

Weisberg and Alba had thought that insight problems are difficult because people are conditioned to think within implied boundaries. How-

ever, even with hints to draw outside of the (implied) boundaries, less than 25% of the subjects solved the problem.

Baker-Sennett and Ceci showed subjects images and words with varying amounts of information omitted. Those subjects who made the 'leap' and identified the image or word did better in the insight problems like the 9-dot problem.

Alex Osborn's early work on creativity, *Applied Imagination* (Osborn, 1953), introduced a three-stage model of the *Creative Problem Solving* (CPS) process. The three stages were fact-finding, idea-finding and solution finding. Sidney Parnes, writes Daniel Couger (1996, p. 86) updated the model in 1967 adding two stages to highlight the importance of problem-finding and acceptance-finding (Parnes, 1967). This model was in turn revised by Isaksen and Treffinger (1985, p. 16), who wanted to emphasise that many creative problems are poorly defined (as discussed above), and added a preliminary stage of "mess-finding". Their model is redrawn in Figure 4.

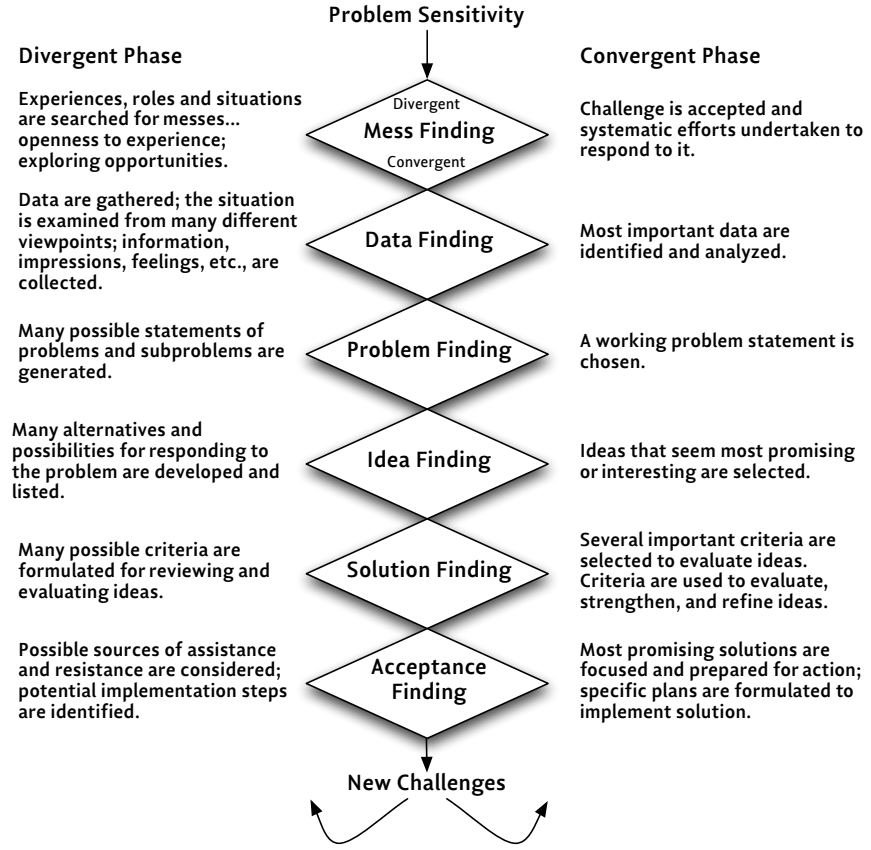


Figure 4. Redrawing of Isaksen and Treffinger's *Creative Problem Solving Process.* (Scott G. Isaksen & Treffinger, 1985, p. 16)

Their model continues to be refined and elaborated as creativity research progresses (S. G. Isaksen & Treffinger, 2004).

An important aspect of these models is divergence-convergence thinking (first introduced, says Couger, by Guilford (1950)). Each stage of a CPS has divergence phases, where many ways to progress through the stage are considered, and a convergent phase, where a particular way of progression is chosen. For example, Isaksen's and Treffinger's CPS model's 'solution-finding' stage has a divergent phase described as "many possible criteria are formulated for reviewing and evaluating ideas" and a convergent phase described as "several important criteria are selected to evaluate ideas. [These] criteria are used to evaluate, strengthen, and refine ideas." (Scott G. Isaksen & Treffinger, 1985, p. 16)

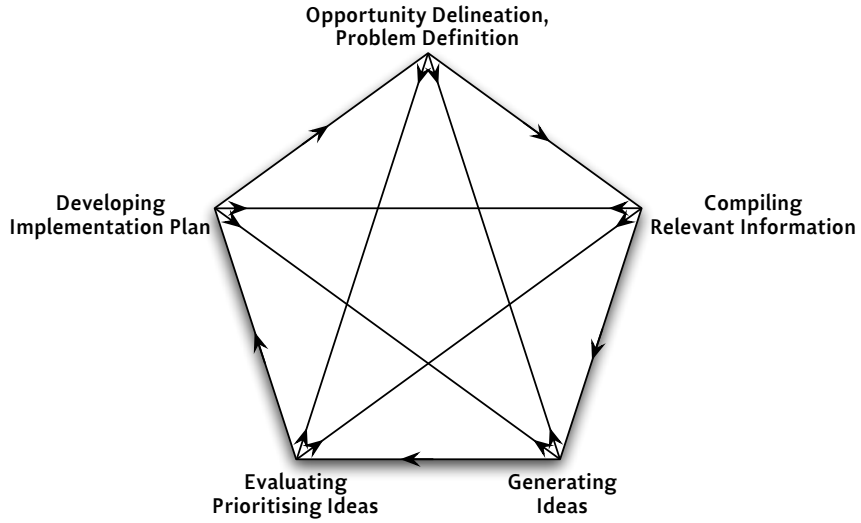


Figure 4. Redrawing of Couger's Creativity Problem Solving process for information systems.
(Couger, 1996)

Couger himself developed a variant CPS model specific to information systems. Each of his (five) phases (shown in Couger's alternate arrangement in Figure 4) is allied with several of 22 'creativity techniques', which are a representative set of heuristics for geminating creative ideas. Each technique can assist with more than one stage of the creativity process, though legibility precludes inclusion of every technique under each activity stage in my Qualities of Creative Environments diagram. The 22 techniques are:

- *Attribute Association* (list the attributes of the solution)
- *Analogies / Metaphors*
- *Boundary Examination*
- *Bug List* (identify irritations)
- *Brainstorming* (in a group)
- *Brainwriting* (as individuals)
- *Crawford Blue Slip* (ideas are written on slips by individuals, and arranged as a group)

- *Disjointed Incrementalism* (Determine the policies involved in the analysis; break the policies into its increments, or stages; evaluate each policy)
- *Decomposable Matrices* (construct a matrix which sets each potential solution against each problem aspect, then assign a rating to each of the interactions)*Interrogatories* (Ask Who? What? Where? When? Why? Then How?)
- *Force-Field Analysis* (describe the 'forces' tugging the present situation to the worst possible, or best possible situation)
- *Goal/Wish* (identify a goal, use concepts from a distant field to generate unusual ideas)
- *Lotus Blossom* (peel back assumptions about a situation)
- *L/R Brain Alternations* (consciously focus on each hemisphere's modality to arrive at a holistic solution)
- *Morphological Connections* (three dimensions of a problem are chosen, and random positions within this 3-space are evaluated in order to narrow down alternatives)
- *Manipulative Verbs* (Manipulate the problem in several particular ways, e.g. Divide, Add, Invert, Transpose)
- *Nominal Group Technique* (generate ideas individually, discuss as a group, iterate)
- *Progressive Abstraction* (for identifying underlying problems)
- *Problem Reversal*
- *Peaceful Setting*
- *Wild Idea* (generation of impractical ideas in order to inspire unusual solutions)
- *Wishful Thinking*

(Couger, 1996, pp. 247-273)

Ben Shneiderman (2002a; Shneiderman, 2002b, p. 83), influenced by Czikszentmihalyi's social model of creativity (see p. 26), as well as Couger's 22 creativity techniques, adopts a simplified framework with these four activities:

- *Collect*: Learn from previous works stored in libraries, the Web, and other sources
- *Relate*: Consult with peers and mentors at early, middle, and late stages.
- *Create*: Explore, compose, and evaluate possible solutions.
- *Donate*: Disseminate the results and contribute to libraries, the Web, and other sources.

(Shneiderman, 2002a)

Shneiderman goes on to propose eight specific tasks, repeated application of which 'should help more people be more creative more of the time'. These are:

- *Searching* and browsing digital libraries, the Web, and other resources
- *Visualising* data and processes to understand and discover relationships
- *Consulting* with peers and mentors for intellectual and emotional support
- *Thinking* by free associations to make new combinations of ideas
- *Exploring* solutions—what-if tools and simulation models
- *Composing* artefacts and performances step-by-step
- *Reviewing* and replaying session histories to support reflection
- *Disseminating* results to gain recognition and add to the searchable resources

Shneiderman, like Couger before him, emphasises that these tasks are not a complete set, but that "they may be helpful in analysing existing software and in designing new tools" (Shneiderman, 2002a). Most, if not all, of these tasks represent open creativity support research questions. To take 'searching' as an example, despite the success of Google, the search problem is a long way from being solved. "As one researcher puts it, the retrieval job is worse than looking for a needle in a haystack; it's like looking for a specific needle in a needle stack" (Dreyfus, 2001, p. 12). Dreyfus goes on to characterise the differences between data retrieval and document retrieval, reproduced in Table 3.

Data Search	Document Search
I want to know x	I want to know about x
Necessary relation between a request and a satisfactory answer	Probabilistic relation between a request and a satisfactory document
Criterion of success is correctness	Criterion of success is utility
Scaling up is not a major problem	Scaling up is a major problem

Table 3. Drefus' comparison of searching for data and searching for documents. (Dreyfus, 2001)

This table shows an example of the problems facing creative workers when dealing with technology: they are not dealing with the technology in terms of discrete components, of rights and wrongs, but instead are wanting to make far more holistic statements much more at the level of meaningful human cognition.

In research by Linda Candy and Ernest Edmonds, empirical studies were used to identify some examples of aspects of creative exploration: Breaking with convention, immersion in the activity, taking a holistic view, parallel channels of exploration (Candy & Edmonds, 2002, p. 71). This list is incomplete, like Shneiderman's, but being based on observations of art practice, it has a closer alignment with the personality traits of highly creative individuals, described earlier. In implicit support of *immersion*, for example, artist John McCormack wrote "Instantaneous-

ness of results (via interaction) means immersion. … the new method provides an intellectual and visual fluid for us to swim in" (McCormack, 1994), and we might take this fluidity as a metaphorical link between *immersion in the activity* and *situated action*.

Candy and Edmonds go on to list some activities involved in the creative process. The list is: ideas generation, problem-finding and formulation, applying strategies for innovation, acquiring new methods or skills and using expert knowledge. Candy and Edmonds comment that digital artists are most interested in the last two of these. Ullman et al. (Ullman, Dietterich, & Stauffer, 1988) describe the Task/Episode Accumulation model of mechanical design, based upon think-aloud studies of designers working on simple design problems. The key features of the model are:

1. the design is constructed by incrementally (iteratively) refining and patching an initial conceptual design
2. design alternatives are not considered outside the boundaries of design episodes (short stretches of problem solving aimed at specific goals)
3. the design process is controlled locally, primarily at the level of individual episodes.

Guindon & Curtis (1987) conducted a similar study on software designers, and identified various design process control strategies, most prominently an opportunistic strategy, where designers move between the problem domain and the solution domain, at high and low levels as they see fit.

Both Guindon & Curtis', and Ullman et al.'s models both relate to what Amabile (1996) called "Algorithmic" problems (as opposed to "Heuristic"), with specified constraints, which, says Amabile, are not always conducive to creativity. However, it will become relevant later to note that in both cases the designers did not use a top-down or bottom-up approach, instead incrementally refining sketches that incorporated both high- and low-level elements.

CREATIVITY SUPPORT QUALITIES OF COLLABORATION

Amabile (Amabile et al., 2001) notes from the literature three categories of team and team member characteristics that appear to predict collaborative success, and augments these with a new study of collaboration between academics and practitioners:

1. Project-relevant skill and knowledge, most importantly diversity and complementary skills, perspectives and knowledge, paired with a common core of understanding about the problem domain

2. Collaboration skill, which stems from experience in collaborative relationships. Amabile adds compatibility of problem-solving styles and leader skill in managing collaboration.

3. Attitudes and motivation, as detailed earlier in this review, including trust, intrinsic interest and openness to change and new ideas, and understanding of possible cultural differences.

These personal characteristics of collaborators are reflected in the processes that those collaborators go through. Amabile identifies, from the literature, positive and negative conflicts, which can result in creative solutions or disagreements depending on the process of resolution. Collaboration appears to be more successful when groups have high levels of cooperative interaction and information-sharing, strong processes for coordination of activities, mechanisms for developing a shared vision for their group project, and when there is a clear understanding about their roles and responsibilities.

It seems to me that this view of collaborative success is at odds with the literature on creative success, most notably that creative people prefer to isolate themselves, and that the roles and responsibilities of creative works is likely to shift as the problem shifts. This would imply that creative collaborations are less likely to be successful, or are harder to manage, than 'non-creative' (for example, goal-oriented) collaborations. Perhaps collaborative skill in creative collaborations must involve the ability to 'put up with' creative personalities and activities.

Amabile suggests that organisations which support collaboration are likely to strongly value people and productivity, and her new research

into the characteristics of collaboration across organisations suggested that effective use of member capabilities and frequent meetings planned well in advance facilitated the functioning of the collaboration, particularly for geographically-dispersed teams.

Gerhard Fischer's "Creativity and Distributed Intelligence" section of the NSF *Creativity Support Tools* workshop report (Fischer, 2005b) expounds the differences between individual creativity and social creativity and emphasises that the two modes should be coordinated and integrated, not opposed. Individual creativity, writes Fischer, comes from the unique perspective that an individual brings to bear on a given problem. Social creativity takes place in a community where different individuals have a shared understanding that supports collaborative learning and working. The community can be distanced and diverse in many different ways (technologically, conceptually, spatially and temporally, for example), and there is the opportunity for tools to bridge these boundaries.

Fischer then describes some qualities of a proposed 'enriched framework for creativity' within this socially-integrated context, namely 'externalisations', 'metadesign', 'from reflective practitioners to reflective communities', and 'from given tasks to personally-meaningful activities'.

Of these four, metadesign is most relevant to my research, and is founded on 'the observation that creativity requires open systems that users can modify and evolve' (Fischer, 2005b, p. 72). Fischer had expanded the concept in a journal paper (Fischer, Giaccardi, Ye, Sutcliffe, & Mehandjiev, 2004). His analysis shows that metadesign allows participants to 'transcend beyond the information given to incrementally acquire *ownership* in problems', and that there is a 'shift in focus from finished products or complete solutions to conditions, contexts and tools'.

It seems to me that the other three dimensions share this focus on ownership (as a means to promote engagement, described by Nakakoji and dealt with below), and the means to develop and share that ownership. 'Externalisation' of creativity is a key component of this process, as boundary objects and as aids to individual reflection. In communities, it

seems that the goal of Fischer's framework is to develop and share ownership of overlapping areas of creativity; so ownership is both individual and social.

Fischer's observations about metadesign sum up a thread which is woven throughout creativity tools research, that because creativity is such an unconfined quality, changing in nature from situation to situation, creativity support tools must be similarly supple, able to adapt with the situation, or if not possible, to relinquish locus of control easily. This freedom and flexibility affords tool-users senses of personal and group ownership, embodiment, engagement and trust, and hence empowers them to work meaningfully and creatively.

Fischer's research into computational support for complex design problems involving the interaction of multiple knowledge systems, offers the following description of collaboration: collaborators come together from different 'Communities of Practice' to form a 'Community of Interest' (Fischer, 2001). Communities of Interest have greater potential for creativity than Communities of Practice (Fischer & Ostwald, 2003), because different backgrounds and perspectives can lead to new insights. Fischer describes the fundamental challenges facing a Community of Interest as being building a shared understanding, learning from and communicating with others with different perspectives. This communication and learning is more complex and multifaceted, says Fischer, than 'peripheral participation' within a single discipline—the work is multidisciplinary.

Fischer goes on to describe Domain-Oriented Design Environments, made during the 1990s, which support a single domain, or Community of Practice. The achievement of Domain-Oriented Design Environments is "external simplicity with internal complexity" each individual has to bridge only a short conceptual distance between problems in their domain to the design environments for their respective domain. Fischer points out that such environments have insufficient support for collaborative design.

CREATIVITY SUPPORT QUALITIES OF ARTEFACTS

Any software that allows users to store and retrieve information could conceivably have creative applications, and a complete review of all of

these is unfeasible. For this section I shall concentrate on the properties of software tools specifically designed to support creativity, i.e. to help knowledge workers manipulate problems. We have already seen Shneiderman's work in this area (see p. 41) who notes "the widespread desire to produce scientific papers, patient treatment plans, legal briefs and popular songs means there could be a substantial audience for effective tools" (Shneiderman, 2002a, p. 116). A note of caution, first: like any tool, software has its effect on the 'look and feel' of the result. Stephen Wilson talks about problems with using computers as creative tools: "The computer is not a neutral object … the computer screen and its image conventions derive from a long history of representation in Western culture stretching from painting through perspective, photography, and cinema, to computer animation and desktop metaphors" (S. Wilson, 2002, p. 729). Such effects are often mutable, but often not without subverting the original function of the tool (Jennings & Giaccardi, 2005). In addition, artistic experimentation is so quickly assimilated (S. Wilson, 2002, p. 10), particularly with computer technology, that it becomes easier for people to reuse existing technology than to create their own. This explains the visual qualities of programs like Photoshop, an effect so recognisable that the tool's name is used as verb ("That image has been Photoshopped").

> *[The shared] type of learning in Communities of Interest requires externalisations [Bruner, 1996] in the form of boundary objects [Star, 1989] which have meaning across the [spatial, temporal, conceptual and technological] boundaries of the individual knowledge systems. Boundary objects allow different knowledge systems to interact by providing a shared reference that is meaningful within both systems. Computational support for Communities of Interest must therefore enable mutual learning through the creation, discussion, and refinement of boundary objects that allow the knowledge systems of different Communities of Practice to interact*
>
> (Fischer, 2001, p. 4).

Boundary objects are physical—they can be pointed to, making sure that collaborators know they are referring to the same thing. In a later book chapter, Fischer writes that boundary objects should be

> *…conceptualised as evolving artefacts that become understandable and meaningful as they are used, discussed and refined. For this reason, boundary objects should be conceptualised as reminders that trigger knowledge, or as conversation pieces that ground shared understanding, rather than containers of knowledge. The interaction around a boundary object is what creates and communicates knowledge, not the object itself.*
>
> (Fischer, 2002, p. 12)

In their study of the design processes of software designers, Guindon & Curtis (1988) suggest some more implications for tools to support designers. They list:

- Library of reusable design schemas, which would provide a decomposition of a problem into sub-problems. This is related to design patterns for particular tasks.
- Design Journal. This would provide an agenda of goals and a list of outstanding issues, reducing the management burden on creative people (or their managers).
- Special displays of constraints in order to augment working memory.
- Visual simulation tools. "Graphics, animation and informal representations need to be incorporated in executable specifications languages to support a smooth transition from upstream design activities to formal specification and implementation"

(Guindon & Curtis, 1988, p. 268).

Guindon & Curtis' recommendations are in alignment with Ullman et al.'s (1988) recommendations for CAD systems (which they cite). The latter suggest that:

- CAD tools should be designed to give cognitive support to the developer, and presumably as part of this,
- CAD tools should be extended to represent the state of the design at more abstract levels
- CAD tools should help the designer manage constraints

In their section of the Creativity Support Tools report, Mitch Resnick and his eminent colleagues describe, with examples and references from the literature, 12 "Design Principles for Tools to Support Creative Thinking" (M. Resnick et al., 2005). These are (with my notes):

1. Support exploration
2. Low threshold, high ceiling, and wide walls. This means it is easy for novices to get started, but that the tools are sophisticated, and suggest a wide range of explorations. This is a rephrasing of Alan Kay's "Simple things should be simple; complex things should be possible".
3. Support many paths and many styles
4. Support collaboration
5. Support open interchange. Seamless operation with other tools that the creative user may wish to employ
6. .Make it as simple as possible—and maybe even simpler. It is easy to sell a product which contains 'more features', and gatekeeper 'experts' are used to using complex tools, say Resnick et al. However, in their experience, reducing the number of features can actually improve the user experience. I find this an intriguing point to make, given the psychologists' characterisation of creative people being attracted to complexity, described previously. The nature of such attraction, in relation to tool simplicity, can be viewed in three ways: the first is that the attraction is hinged upon the possibility of simplifying the complexity (for example, by discovering a law which governs a complex behaviour). Or conversely, it could be that creative people wish to build complexity upon simple foundations. A third, more sensible, possibility is that this

simplification/complexification process is fluctuating or iterative, recalling Guilford's divergent/convergent thinking.

7. Choose black boxes carefully. The choice of the 'primitive elements' of a tool is extremely important, because they determine what ideas the creative user can explore with that tool.

8. Invent things that you would enjoy using yourself. If the tool's designer doesn't enjoy using the tool, why would other users? There is a risk here of a designer designing only for his or her own use; but observation and tool evaluation, below, counteract this.

9. Balance user suggestions with observation and participatory processes. Users do not always know what they want. Whilst their suggestions may often be helpful, they may, for example ask for too much flexibility, or for impractical features. On the other hand, they may not have the perspective to be aware of possibilities for radical changes.

10. Iterate, iterate, then iterate again. A cyclic process of requirements gathering, implementation and evaluation refines the design.

11. Design for designers (which Fischer calls *metadesign*). Equip users with simple tools so they can express themselves creatively (conversely, don't equip them with complex tools which limit their expression).

12. Evaluation of tools. "It is still an open question how to measure the extent to which a tool fosters creative thinking. While the rigour of controlled studies makes them the traditional method for scientific research, longitudinal studies with active users for weeks or months seem a valid method to gain deep insights about what is helpful (and why) to creative individuals" (M. Resnick et al., 2005, p. 34).

Whilst researchers such as Brock Craft (Craft & Cairns, 2005) warn of the dangers of blindly following guidelines and assuming that following guidelines automatically makes for an appropriate product, these guidelines are useful as topics to consider when evaluating the design and

product. When considering their inclusion in my Creativity Support Qualities diagram, three of these design qualities (1, 3 and 4) are support for activities and thinking styles, which are already included by the diagram, and the other principles are added.

In the same NSF report, Kumiyo Nakakoji describes "Seven Issues for Creativity Support Tool Researchers". Whilst these are not the only issues that this community faces, several important concepts are raised. These are:

1. *Three roles of tools supporting creativity* (see below). Nakakoji draws analogies between these roles in creative practice and the roles of dumbbells (good for skills development and maintenance), running shoes (easing an existing experience) and skis (provision of new types of experience) in sporting practice. Whilst appealing as a metaphor, I'm not sure that the 'dumbbells' role, the cognitive development role, is the same type of role as the other two, which are based upon augmenting experience. These other two roles seem to lie on a continuum, seemingly of 'novelty of experience'. To paraphrase Nakakoji, the experience of skiing cannot take place without using skis. Similarly, the experience of running-in-running-shoes cannot take place without using running shoes, and running-in-running-shoes is quite a different experience to running without, not just 'easier'. Even lifting a dumbbell for a fourth time can be a different experience to lifting it for the third time. Every change provides a new experience; the distinguishing factor from a creativity support point of view is the novelty of the experience provided. Whether this new experience is novel enough for us to achieve something that we were unable to achieve before depends on the situation and an individual's application of the gained experience. So, since novelty is a widely-cited concurrence of creativity, we can say that given 'appropriateness to the situation', the more novel a tool, the more likely it is to stimulate creativity.

2. *Engagement, Embodiment, Trust and More*. Creativity support tools need to promote engagement, embodiment and trust in their users, and that the utility of tools which do so may not be described

in terms of productivity and efficiency, but of more subjective ones, such as aesthetics, flow, preference and value. Nakakoji states, "as researchers, our goal is to develop rigorous accounts and identify scientific evidence for how such concepts are instantiated within tools", and makes the point that new media art works often instantiate some of these concepts, and hence are worthy of study from this point of view.

3. *Creative Processes in Software Development*. Nakakoji argues that software development (a representative creative activity) research needs to take more into account of the individual differences in programming process. Extreme Programming (XP) offers a free-form programming style which outperforms more structured styles. The success of open source development has also demonstrated the importance of studying social creativity.

4. *Cabinets of Curiosity.* Tools should support the 'collection' phase of Shneiderman's model of creativity.

5. *Learning from Media Arts* (see above).

6. *Creativity in Context*. Different tools are applicable in different contexts, and, firstly creativity researchers must be clear about the context in which the research operates, and secondly, tool designers need to ease the transition between different tools (as identified by the "open interchange" guideline above).

7. *Beyond Adaptation and End-User Modifiability.* Nakakoji contends that "adaptive mechanisms and end-user modifiability have been explored as ways to allow people to adjust tools for individual differences, but such minor adjustments cannot afford the variety of tool needs." This is indeed the case with current technology, but my counter-contention is that if end-user modifiability is *sufficiently flexible*, i.e. as flexible as a programming language, it would be possible to make any change that a programming language is capable of, and its use would no longer be the sole domain of the end user. Obviously this flexibility would require a radically-different architecture for building end-user-modifiable systems.

CREATIVITY SUPPORT IN BALANCE

One thought to finish with. Mark A. Runco and Shawn Okuda Sakamoto, summing up their review of experimental studies of creativity, agree with Nickerson's "balance" quality, and observe that:

> *When important determinants or contributions to creativity are identified, it is often the case that the benefit is greatest at an optimal (not maximal level). This holds true of information, experience, arousal, conflict—and probably much more […] too much of just about anything tends to be detrimental.*
>
> (Runco & Sakamoto, 1999, p. 80)

Interactive Art

As I wrote in the introductory chapter, I define interactive art to be any art that responds to the presence of, or action by, participants in the artwork's audience. The very use of the terms audience and participants indicate that the role of an artwork's public is very different to the traditional roles of art audiences as passive or even active consumers.

In 1973 Stroud Cornock and Ernest Edmonds proposed that rather than talk about 'artworks,' it was helpful to think in terms of 'art systems' that embraced all of the participating entities, including the human viewer (Cornock & Edmonds, 1973). These systems were categorised into four types, namely static, dynamic-passive, dynamic-interactive [unvarying] and dynamic-interactive (varying). ([Unvarying] is my addition.) In the light of developments since that paper, we can reconfigure these categories to show us more about the interactivity of current styles of art.

Cornock and Edmonds use the terms 'static' and 'dynamic' to connote temporal change rather than change in what is contained within the art system. A painting is therefore static, and a cartoon is dynamic-passive. A dynamic-passive work changes as time passes, often because of computer or even another human. If a participant is the agent for change in his or her own experience of the work, the work is interactive. However, some confusion arises when the perception of the work (as opposed to the content of the work) alters, for example through navigation around it, or gaining of experience.

This idea can be extended by considering feminist physicist Karen Barad's neologism "intra-action" (Suchman, 2003, p. 13) emphasising that "subjects and objects emerge through their encounters with each other". Barad goes on to say that technoscientific practices are critical sites for the emergence of new subjects and objects. Suchman uses the irreducibility of these encounters in another paper (Suchman, 2002) to show that it is from the affective encounter that the most exciting developments in interaction are coming.

A case in point is Jeffrey Shaw's and David Pledger's *Eavesdrop* (Shaw & Pledger, 2004). The content of the artwork is a looping 360-degree video of a stage, in which various sub-plots are acted out. The navigation of this work is a participant-controlled rotating turntable, constraining the 'eavesdropping' view to only one sub-plot at a time.

Now, the content of *Eavesdrop* is dynamic-passive—nothing a participant can do affects the repeating story sequences; all the participant can do is alter the order in which he or she watches them by rotating the turntable. In itself, this isn't fundamentally different to walking around a traditional kinetic sculpture in one direction or other. Yet it is this navigation that is fundamental to the piece—indeed, it is what brings meaning to its name; and since changing the navigation route changes what an inexperienced participant takes from the piece, we are compelled to say that the piece is dynamic-interactive (unvarying), or functionally indistinguishable from it. However, once a participant has exhausted all of the narratives, and repeated use of the piece conveys no new information to him or her, we could say that the piece reverts to being dynamic-passive or even static—the piece does not change for that participant no matter what he or she does.

Another example, which employs game technology, is *acmipark* (Oliver, Chatterton, Blundell, & Cannon, 1998), a multi-user environment modelled upon the ACMI (Australian Centre for the Moving Image) building in Melbourne. Assuming (naïvely) that the artwork is the virtual model, and the means by which it is navigated round it is irrelevant, *acmipark* is static. However, a participant could encounter other participants, or their avatars, in the unchanging space. These other participants could be viewed themselves as changing the space by their very pres-

ence, thus creating a dynamic-passive space. Further, non-repeatable interaction between a given participant and the others in the space could make the space dynamic-interactive (varying). The lesson here is that Cornock's and Edmonds' classification varies according to what is included in the space of an art system (in these example, whether the content includes the method of navigation, or any other participants), and also according to how participants' engagement changes over time.

However, many artworks are unambiguously interactive. The participant objectively changes the space simply by being a participant. Thus, over time, the space becomes permeated by the participant (or participants, if the changes are retained or disseminated), and the participant can observe these changes. This gives the participant senses of agency, contribution to and ownership of the artwork and their experience of it. In return, the artwork can show participants 'transformed mirror images' of themselves (Rokeby, 1996) and others, making the experience unique and customised to each participant.

A third kind of space to navigate is a space constructed *from* the participant him- or herself—the gestalt space does not manifest without him or her. Whether this is fundamentally different to the second kind of space or not is a philosophical and critical question, and it does not matter for our purposes, but the result is that the participant 'owns' the space (the space may even become an extension of the participant), and that navigation of the space becomes a new way to navigate round the participant's self. These works seem to be comparatively rare, but an excellent example is George Khut's *Cardiomorphologies v. 2*, of which we will hear much more in Chapter 8.

The histories of art and of technology are well intertwined. Stephen Wilson, in his encyclopaedic review of science in modern art, states:

> *We are at an interesting place in history, in which it is sometimes difficult to distinguish between techno-scientific research and art—a sign that broader integrated views of art and research are developing.*
>
> (S. Wilson, 2002, p. 4)

Figure 5. Futurism, originating in Italy in 1909, was preoccupied with modern machinery, transport and communication. (Luigi Rissolo, Dynamism of an Automobile, 1912-13, oil on canvas.)

In particular, audience-involvement art and technology-enabled art share predecessors, particularly in the form of what is known as 'process

art'. The relationship of computers to art is prefigured by one of the most enduring ideas in art in the past hundred years—that of incorporating new technology into art. This connection with science and technology is representative of artists' desires to engage with the increasing presence of these factors in the physical and social environments around them. Such desire was first made explicit by the Futurists, and their preoccupation with technology (Figure 4) which influenced the Russian Constructivists' uses of industrial processes to make new imaginative forms (Figure 5). Many constructivists, for example László Moholy-Nagy [1] (Figures 6 and 7), went on to teach at the Bauhaus school of art and design, where the design approaches were more rigorous, and the goal was to produce universal designers able to work with equal creativity in the fields of architecture, handcrafts, or industry.

Figure 6. This piece is one of many which Moholy-Nagy described as 'systems'—perhaps a foresight of Cornock and Edmonds' description of art as systems (p. 53)
(László Moholy-Nagy, Kinetic constructive system, 1922, watercolor, Indian ink, collage.)

Whilst this line of descent is the most relevant, there are other more disparate examples of artists' preoccupation with technology. Man Ray's 'solarisations' and 'rayographs' (Figure 8) were innovative artistic techniques resulting from experimentation with technology, and Ray made a large contribution to the change in perception of photography as an art form rather than a science[2] . Marcel Duchamp's 'readymades' introduced the notion that art did not necessarily have to involve beauty and artisanship; simply "choosing" manufactured articles with appropriate conceptualisation was sufficient.

In the field of sonic art, things are no different. Today's preoccupation with technologically-mediated sounds (sampling, synthesis, turntablism, effects and processing) can be traced back to, for example, Pierre Schaffer, pioneer of musique concrète (e.g. Étude aux chemins de fer (1948), which was made from recordings of trains) or to John Cage (whose father was an inventor of almost useless devices[3] , who wrote music for

Figure 7. In this culmination of Moholy-Nagy's ideas, his plan was to project light onto a blank wall through the moving parts of a machine constructed of metal and glass elements and of new materials like plexiglass, thus creating arresting patterns from the intermingling bursts of light and shadow, which are reminiscent of his still photographs.
(László Moholy-Nagy, Light-Space Modulator, 1930, kinetic sculpture.)

1 With characteristic prescience, Moholy-Nagy speculated in 1930 on the networked distribution of the Light-Space Modulator: "It is indeed foreseeable that this or similar motion pictures may be transmitted by radio. In part with the support of telescopal prospects, in part as real light plays, whereby the listener owns a private lighting apparatus which can be remotely conducted from the radio station via electrically controlled colour filters" (Droste et al., c. 2000).

2 This change in perception was exemplified when Man Ray was described by an artist neighbour as "the photographer who thought he was a painter" (Baldwin, 1988)

3 http://en.wikipedia.org/wiki/John_Cage

Radio Receivers (Imaginary Landscape No. 4 (1951)). Later, and influentially, Steve Reich's tape-looping pieces It's Gonna Rain (1965) and Come Out (1966), used systematic manipulation of technology to reveal the rhythmic and tonal elements in the spoken word. Returning to Cage—a year after Imaginary Landscape No. 4, he premiered the infamous 4′33″, a piece in which the performers remain silent, but in which the sounds of the audience and environment may be considered to be the music. 4′33″ marked a brilliant breaking of traditional audience/performer/composer constraints in Cage's work, which he pursued for the rest of his career[4].

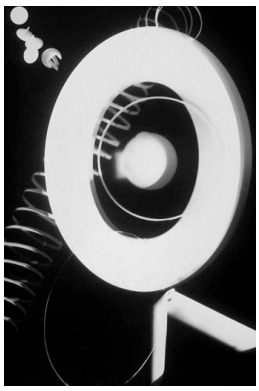


Figure 8. Ray's 'Rayographs' were made by exposing photographic paper on which various objects had been placed. (Man Ray, 1922, Rayograph, Gelatin Silver Print.)

Taken together with his technology (and process) preoccupation, Cage's involvement of the audience in his work marks him as one of the first producers of interactive technology-enabled art. The systematic and process-led elements in his works are also a hallmark of the Fluxus movement in the 1960s and '70s. The performance of such processes, called 'happenings', was a natural environment in which to involve and respond to audience members. The Fluxus movement reconceptualised art as detached action or process, and one result is that the audience is able to experience or influence the work in the same way as any other enactor of the process (such as the artist). A succinct example of the effect of a Fluxus piece is my execution of Ken Friedman's score, The Distance from This Sentence to Your Eye is My Sculpture (Figure 9).

the distance
from this sentence
to your eye is
my sculpture

Figure 9. An execution of Ken Friedman's fluxus score, The Distance from This Sentence to Your Eye is My Sculpture. (Friedman, 1971, VII Paris Biennale)

We shall continue our exploration of interactive art by looking at the technology behind a range of pieces, in order to show the mind-boggling diversity of forms that interactive art takes and demands that it makes upon the people and technology involved.

I had originally thought to review art pieces to make a taxonomy of styles of interactive art, but quickly realised that art resists rigorous and detailed categorisation, somewhat by nature—contemporary art doubly so and computer-enabled interactive art trebly so (despite my use of that particular category, reasons for which are described in the introductory chapter). Rather than taxonomies, artworks might better be arranged into 'folksonomies' (Mathes, 2004)—a flat hierarchy in which records

4 Interestingly, Cage's estate is able to collect royalties from attributed performances of 4′33″. See, for example, http://news.bbc.co.uk/1/hi/entertainment/music/3401901.stm

assigned descriptive, equivocal tags, such as 'user portraiture', 'virtual reality', 'biofeedback', and so on. Ian Gwilt et al.'s *Untitled Media* project is an attempt to capture folksonomy information about new media art works[5] . It is hence my intention here to show the qualities that characterise what I consider to be computer-enabled interactive art. I will not attempt to invent 'tags' for these qualities, since (as far as I can tell by observing its short tradition) folksonomy tags seem to be created and assigned in order to ally the record to be tagged to other records in someone's (here, the artist's) subjective community. Rather, I shall attempt to suggest by examples some variations in styles of interactive art that arise from particular means of conceptions, production and presentation.

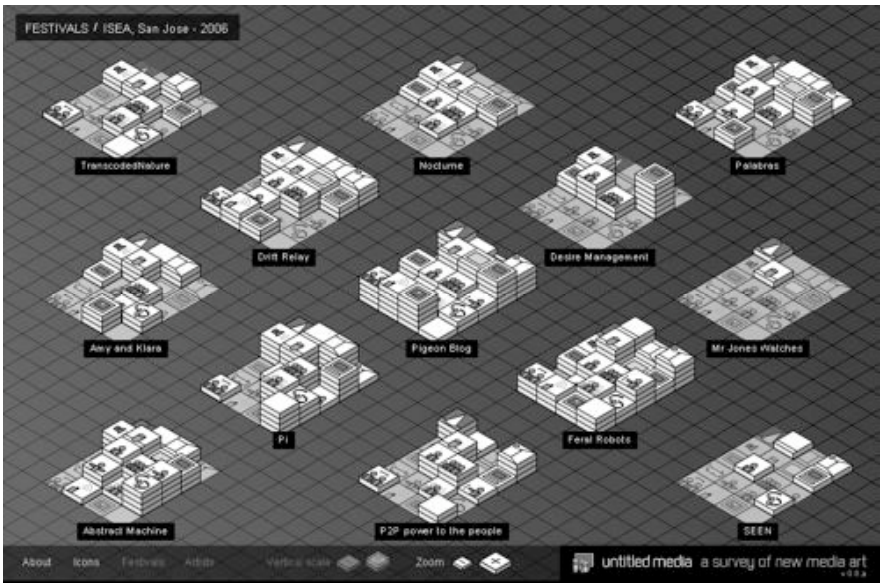


Figure 10. Gwilt et al.'s *Untitled Media*, showing graphics of tags which had been applied to various artworks.
http://www.creativityandcognition.com/UntitledMedia/ UntitledMedia.html

Interactive art has no particular situation—it could take place anywhere. Common spaces to experience it are on the Internet, at galleries and in academic situations, but it is by no means unusual for artists to create art for other public, private and personal spaces.

Art delivered to the personal computer usually interacts on a one-computer-to-one-human basis, or, via the Internet, many-humans-to-

5 http://www.creativityandcognition.com/UntitledMedia/UntitledMedia.html

many-computers-to-many-other-humans. Art delivered in other situations can take on any configuration of humans and computers.

Net art (or '.net art' as some prefer) is perhaps the most common situation in which to experience interactive art, because of the cheap and quick ability to massively distribute it over (usually) the web. Interactive net art is almost universally interacted with using mouse and keyboard on a desktop computer, using a small number of commonly available browser and server technologies, namely HTML (Masayasu, 2005), e.g. jodi.org (Heemskerk & Paesmans, 2005), Adobe's Flash (Adobe, 2005), e.g. Anonymous, (c. 2005) (Figure 11) and Director (Adobe, 2004) and Sun's Java (2005), e.g. works at http://processing.org/. More recently, interactive art has begun to appear on mobile phones, based on the same technologies (except Director). The limitations on the range of hardware and software used to experience net art often reflects the technological lowest common denominator, as supplied by hardware and software providers (namely desktop computers, keyboards and mice, but occasionally cameras and microphones), combined with many artists' desires to be as widely accessible as possible. There have been several exhibitions of net art in physical galleries, complete with mice and keyboards, and even mobile phones, but this approach has the side-effect of removing the art from what might be its intended social situation—as part of an attention-deficit web surfing session (Muller, 2004).

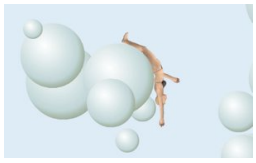


Figure 11. Anonymous (c2005), Never-Ending Fall, a Flash artwork at e.g. http://www.thechump.com/neverendingfall.swf

Internet art has a certain emotional quality of insignificance; of transience. Because of where it resides, outside of any authoritarian hierarchy, and with no tangible existence, and because of how it is viewed, by disembodied anonymous participants, there is a danger that viewers will not invest much time or thought, beyond a fleeting surprise or wonder, in a given piece. Philosopher Hubert Dreyfus writes:

> *Clearly, the user of a hyper-connected library would no longer be a modern subject with a fixed identity who desires a more complete and reliable model of the world, but rather a postmodern, protean being ready to be opened up to ever new horizons. Such a new being is not interested in collecting what is significant but in connecting to as wide a web of information as possible.*
>
> (Dreyfus, 2001, p. 11)

yet he goes on to say,

> *Web surfers embrace proliferating information as a contribution to a new form of life in which surprise and wonder are more important than meaning or usefulness. This approach appeals especially to those who like the idea of rejecting hierarchy and authority and who do not have to worry about the practical problem of finding relevant information. So postmodern theorists and artists embrace hyperlinks as a way of freeing us from anonymous specialists organising our databases and deciding for us what is relevant to what.*
>
> (p. 12)

Desktop interactive art that demands more bandwidth, or more attention from its participants than net art can be supplied in a more self-contained manner, from 'shows' created in tools such as Flash and Director to custom-compiled software e.g. (A. Ward, 2002). Such art will often take over the computer, in order to have the participant's undivided attention. Desktop interactive art (the desktop being the most intimate and personalised location for interactive art) has the luxury of being able to require and reward interaction over extended periods. Participants are able to come acquainted with a piece's depths to a greater extent than over the net or in a gallery.

Galleries and appropriate academic spaces offer an opportunity for interactive art to be greatly refined in presentation and environment, due to the highly controlled and flexible nature of such spaces, and the great effort it is sometimes possible to make to install a piece. It is not unusual, for example, to use unconventional sensors and displays, and many separate computing devices (Figure 12). Additionally, through the curatorial process, an installed artwork can be surrounded by various artefacts (including lighting, text, gallery attendants, and other pieces), which are aimed to make engaging with and understanding the piece easier, and more rewarding.

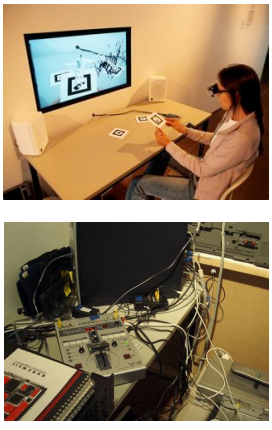


Figure 12. My photos of frontstage and backstage of Rodney Berry's *In the Domain of the King* showing many separate devices being connected to produce the integrated artwork.

Another potential place for interactive art is in private dwellings. Whilst some private companies have been known to commission interactive art, this is relatively uncommon. Because of new media art's demanding preservation requirements (and interactive art's dark secret of unreliabil-

ity!), works are unpopular purchases for those without the resources to keep the piece in working order.

The sensors and displays[6] for net art and desktop art are constrained largely to lowest-common-denominator hardware. Mouse, keyboard, small screen and speakers dominate (though the art often makes unusual use of them). Occasionally microphones are employed. An emerging real-time desktop sensor is the webcam, as the popularity of such devices increases.

Some interactive net art requires or allows participants to supply their own media—pictures or sound, usually created 'off-line' and introduced to the system, occasionally via still camera or microphone (e.g. http://homokaasu.org/rasterbator/). These media are often stored and shared with other users (e.g. Smith and Hagen's S*ocialising with Strangers (*K. Smith, 2004)). For the purposes of this thesis I have limited consideration to devices that supply and react to information in real-time (not, for example, still cameras or medical testing). This is because real-time interaction is most challenging to design for, and satisfying requirements for real-time are likely to satisfy the requirements for non-real-time interaction.

For installed art, the range of sensors and displays is virtually unlimited—any interaction device that can be connected to a computer could be used in interactive art. Devices used in installations often have the effect that they disturb the traditional boundaries between computer and participant, in two significant ways: by overriding human senses, often in an invasive way (à la Virtual or Augmented Reality) or by reducing the physical presence of the interface (particularly in ambient pieces). A seminal example of the former kind of interface can be found in Char Davies' Osmose (Davies, 1995) (Figure 13), in which the participant dons a virtual reality headset and a vest that senses breathing. The participant is able to control buoyancy by breathing. In Davies' statement, she writes:

Figure 13 Screenshot from *Osmose*
(Char Davies, *Osmose*, 1995, Virtual Reality Installation)

6 'displays' is used here in the sense of 'a human interface device for output', not just video displays.

> *In contrast to the hard-edged realism of most 3D-computer graphics, the visual aesthetic of Osmose is semi- representational / semi-abstract and translucent, consisting of semi-transparent textures and flowing particles. … The after-effect of immersion in Osmose can be quite profound. … Such response has confirmed the artist's belief that traditional interface boundaries between machine and human can be transcended even while re-affirming our corporeality, and that Cartesian notions of space as well as illustrative realism can effectively be replaced by more evocative alternatives*
>
> (Davies, 2002).

An example of the second kind of interface, with a reduced physical presence, is Sidney Fels' Iamascope (Fels, 1999) (Figure 14), which is an interactive kaleidoscope. Fels and Kenji Mase write:

> *By keeping the user unencumbered, the sense of engagement and intimacy is very high as soon as they enter. … Many parts are required to create the interaction including: balancing computer graphic and sound qualities, providing real-time, responsive feedback in both media, providing the correct mapping type between body and media, and understanding aspects of intimacy with the interface*
>
> (Fels & Mase, 1999).

Figure 14. Members of the public interacting with *Iamascope*
(Fels, 1999, Camera and Projection)

Common examples of real-time sensors and displays are listed in Tables 4 and 5.

Table 4. Common realtime sensors for installed interactive art.

Device	Description
Light Sensors	The most simple form of light sensor is a photoresistor, which lets more current pass as the light hitting it becomes brighter. This can be used to tell whether or not the sensor is covered, either by an object which the participant moves, or by the participant him or herself. The most complex common form of digial light sensor is a camera. Cameras are often aimed at participants, and the image is used either simply as a depiction of the camera's field of view, or as input to computer vision algorithms to track the positions of elements of the image, for instance objects, bodies, or eyes. To detect 3D location, more than one camera is almost always needed, or objects which are specially marked to give an unambiguous 2D representation of their 3D position. Since human bodies produce heat, IR sensors can be used to detect them. IR sensors can be used in simple ways, such as those in burglar alarms, to detect presence or absence, or slightly more complicated ways, to detect things like proximity, or very complex ways, using an IR camera, to isolate human shapes from the background.
Touch Sensors	In terms of hardware, touch sensors can physically connect two pieces of metal as pressure is applied (such as switches and buttons) or change capacitance in response to skin contact (as found in trackpads and motorised faders), or produce a continuously variable response depending on pressure or displacement. Some pressure sensitive devices can detect multiple points of contact—many people or fingers simultaneously. These hardware devices are often embedded in artwork-specific interaction devices, such as a control panel or pressure-sensitive floor.
Sound Sensors	Microphones offer a way for participants to produce sound for use by the artwork. Like cameras, microphones can either be used simply to record stimuli, or attempts can be made to analyse it, for example by using speech recognition. Sound sensors can also be used to detect vibration, such as tapping on a surface. With several such sensors, it is possible to detect the position of the vibration's source. In some cases, a device which emits sound pulses (often ultrasonic) can be moved around, if tapping is not possible, or if continuous tracking is desired.

Device	Description
Position Sensors	Aside from those techniques mentioned above, there are other ways of detecting the position of something. RFID (Radio Frequency Identification) tags can be attached to objects, and the way in which they interfere with an electromagnetic signal can be analysed to figure out the tag's distance from the antennae. Two tags, which interfere with the signal in different ways, can be used to figure out orientation of the object, and temporarily disabling one tag, for example in response to a button-push, can also be detected. A well-known example of RFID in interaction is the sensetable (http://tangible.media.mit.edu/projects/sensetable/) A type of sensor commonly found in Virtual Reality (VR) systems, attached to Head-Mounted Displays and data-gloves, works by detecting the strength of magnetic fields emitted by a fixed source, based on the sensor's distance from that source. Another technique is to use accelerometers to keep a track of speed and hence relative change in position, but these are prone to drift. Various interesting materials for sensing position (or force as a function of position) have been recently developed, notably bend sensors (e.g. http://www.makingthings.com/teleo/cookbook/bendsensor.htm) and stretch sensors (e.g. http://www.robotstoreuk.com/SENSORS+VISION/MerlinStretchSensor/MerStretchSensor.htm).
Rotation Sensors	Aside from potentiometers ('volume knobs'), and the RFID technique outlined above, some devices use gyroscopic sensors to detect changes in angle.

Device	Description
Biological Sensors	Biological sensors are often based on the principles of other types of sensors - for instance, a common method of detecting blood flow (and hence pulse) is to detect the amount of light which passes through the finger or earlobe. Special sensors are listed here. EKG: measures the electrical activity of the heart. It usually requires three electrodes to be attached to the arms. EEG: Brain activity (e.g. Alpha Waves. Requires several electrodes to be attached to the scalp. Galvanic Skin Response: Detects the conductivity of the skin, which is interpreted as an image of activity in other areas of the body. EMG: measures the electrical activity of muscles. Electric Spirometer: A device that measures the rate of flow of breath. These devices, while sensitive, are prone to drift, and have hygiene issues (every participant must use a different tube, and antibacterial filters must be changed regularly). A more hygienic alternative is to strap a force sensor round the ribs, but this is more sensitive to other movements.

Table 5. Common realtime displays for installed interactive art.

Device	Description
Video Displays	These are the most familiar output devices, because most desktop computers use similar technology. It is not uncommon for interactive art to use more than one video display. The main types are as follows: Cathode Ray Tube (CRT): The most common and cheapest form of video display, these are found on most desktop computers, but because of their unappealing bulk, are seen increasingly rarely in galleries, and often mounted behind a hole in the wall when they are. Some CRTs are aesthetically pleasing enough to be exhibited, such as those designed by Apple. LCD panel: De rigeur on laptops and portable devices and increasingly common on desktops as a space-saving alternative to CRTs, these are now being produced in large sizes. LCDs have traditionally been difficult to see in daylight and difficult to see from all angles, though these problems are being overcome as technology progresses. Many LCDs have 'dead pixels', where one of the elements doesn't respond. These are usually hard to detect, but may be distracting for the discerning art viewer Projection: A video image can be projected from the front or rear of a projection screen, depending on space availability and whether or not participant shadows are likely to be a problem. Front projection is the cheapest large-format video display, but an optically corrected rear projection screen can be expensive. Projector bulbs have limited lives, typically 1000 hours, and replacements are expensive, so care should be taken with always-on use. Projected displays are very low-contrast because the screens are white, and typically require a darkened room, though higher contrast 'black screens' are emerging (http://www.cdfreaks.com/news2.php?ID=9984) Plasma Display: A thin, large display format, brighter and higher contrast than projection, once unique in these respects, but now beginning to be superseded by LCD and OLED. Plasma's major issue when used for interactive art is burn-in—if the same or similar image is displayed for too long, the image becomes 'etched' on the screen.

Device	Description
Video Displays (Continued)	LED video displays, such as those seen at sports stadiums, are very large, very bright and very, very expensive. OLED (Organic Light-Emitting Diode) displays are a promising replacement for LCD and Plasma displays. The manufacturing process of OLED displays is more reliable than for LCDs, making it cheaper and easier to produce large displays. What's more, OLEDs can be 'printed' onto flexible substrates, allowing bendable and wearable displays, and have a lower power requirement than LCDs.
Lights	The function of a light is obvious. Various numbers and forms of light can be controlled, from a single LED to the lighting at a large concert. Many individual light elements can be built up to form larger displays, the most complex being colour video displays.
Speakers	Again, this is a common type of display on desktop computers. Interactive art sound systems are more likely to use professional quality audio components, be louder to fill a large space, and to use more speakers to position the sound in 3D space. Similar technology can be used to cause other forms of vibration, such as ripples in water.
Motors	Motors are used for controlling mechanical devices or causing vibration. Simple motors rotate when a current is passed through them. Stepper motors require a timed sequence of applications of current to various parts of the motor to cause rotation. This allows for great precision and speed, and is used commonly in, for example, printers.
Pneumatics	An efficient way to control robots, for example. Pneumatic devices work by using a motorised pump to pump fluid into or out of a piston, which extends or contracts as a consequence.
Biocontrol	realtime phyiscal biocontrol is fairly rare, mostly for ethical reasons. About the only use of it in interactive art is for muscle stimulation.

A special category of 'sensor' is the networked sensor, particularly those that access very large collections of data and information, and those that connect physically-distant computers (and hence people). Art which uses the massive data accessible via the Internet often makes use of the information visualisation techniques to make us more aware of particular aspects of a network. An example of such a use is Golan Levin et al.'s Secret Lives of Numbers (Figure 15)—an applet which visualises the popularity of the numbers 1 to 1 million (100,000 in the online version).

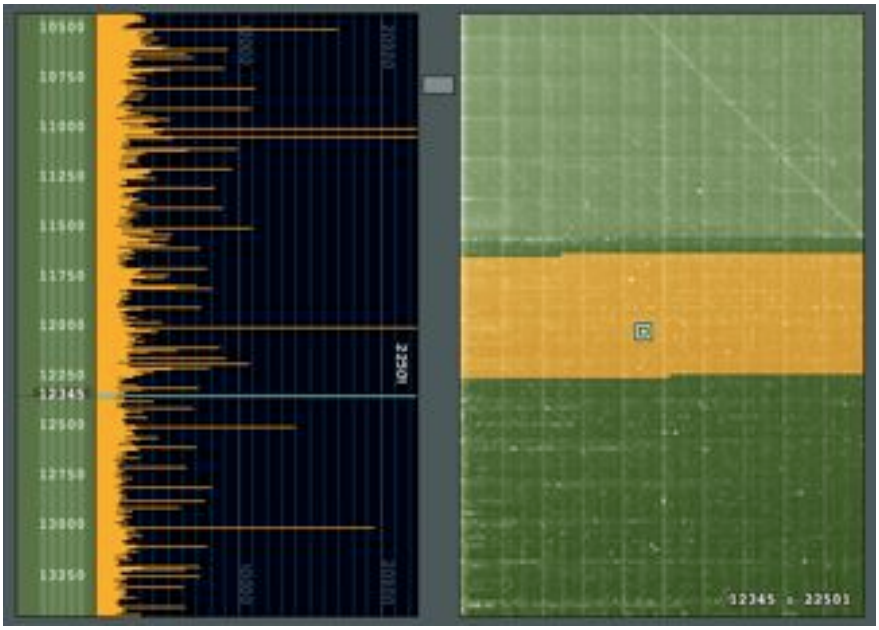


Figure 15. ***The Secret Lives of Numbers***
(Levin et al, 2001)

An example of networking in art can be found in cubeLife (Everitt & Turner, 1999) (Figure 16), described in (E. A. Edmonds, Everitt, Macaulay, & Turner, 2004) and (Everitt, 2002), is an "Internet-Resident Artwork". The gallery version of the piece allows participants to create a unique graphical representations of the structures of magic cubes[7] with their heartbeats. Each magic cube is able to move around a virtual space and over the Internet to interact with other cubes created (with a mouse) by other (online) participants, according to various adjustable rules.

---

7 a magic cube is a cubic array of non-repeating—and in this case sequential—integers in which the n2 rows, n2 columns, n2 pillars, and four 'triagonals' each sum to a single number. The spatial arrangement of numbers has certain intriguing properties.

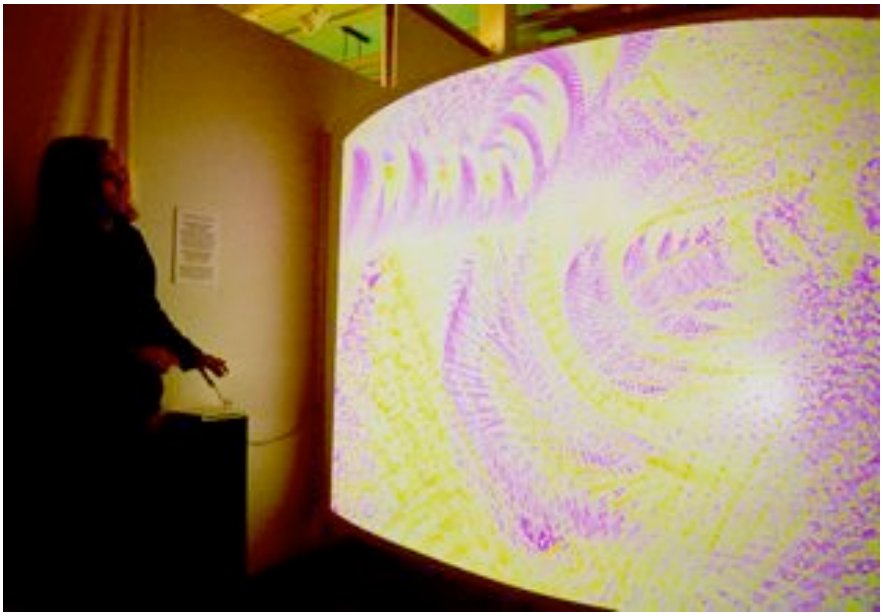


Figure 16. *cubeLife*
(Everitt & Turner, 1999)

PROCESSES FOR CREATING INTERACTIVE ART

Interactive art necessarily involves strong aesthetic and technological elements, and expertise in both is needed in order to be successful. Presented here by way of analogy, Stephen Wilson (S. Wilson, 2002) enumerates some differences between science and art, reproduced in Table 6.

With the possible exception of the last item, the table's headings could be renamed "interactive art" and "computer science". Some of the differences are false dichotomies, intended more to illustrate tendencies than absolutes. What is particularly useful to note are the similarities: crucially, that 'both value creativity'. This shows again not only how diverse a field creativity is, but also that many qualities that (interactive) artists find valuable are shared by (computer) scientists, and vice versa.

Some artists (for example, David Rokeby (e.g. Rokeby, 1996), Casey Reas (e.g. Reas, 2004), Brad Paley (http://www.didi.com/brad/) John McCormack (e.g. McCormack, 1994)) work predominantly alone, bringing all the necessary technical and aesthetic skills to their artworks. Others (e.g. Jeffrey Shaw (e.g. Shaw & Pledger, 2004), Char Davies (e.g.

Davies, 2002), and Alexa Wright[8]) have worked in interdisciplinary collaborations, where artists work with particularly technologists, but also historians, scientists or others to make an artwork that utilises the expertise of each. Whether or not artists should, or do desire to program is a somewhat contentious and recurring issue (indeed, the topic is revisited in Chapters 5 and 6); artists and third-party observers have made various comments about the pros and cons of solo work or collaborations:

	Art	Sciences
Differences	Seeks aesthetic response Emotion and intuition Idiosyncratic Visual or sonic communication Evocative Values break with tradition	Seeks knowledge and understanding Reason Normative Narrative text communication Explanatory Values systematic building on tradition and adherence to standards
Similarities	Both value the careful observation of their environments to gather information through the senses Both value creativity Both propose to introduce change, innovation, or improvement over what exists Both use abstract models to understand the world Both aspire to create works that have universal relevance	

Table 6. Stephen Wilson's assessment of differences and similarities between science and art.
(S. Wilson, 2002)

8 http://www.alteregoinstallation.co.uk/

In the pro-Artist-as-Technologist camp, Tom Hewett (a psychologist and computer scientist) writes in Candy's and Edmonds' book, that

> *One of the key features of creative work is the importance the artists give to the locus of control […]. Being on a voyage of exploration and discovery they want to find their own way. They are typically not interested in being shown a solution unless convinced that the person they are working with understands what they want to accomplish. Working through and with the eyes of the person who provides technical expertise does not work well for the core creative activities although it might be acceptable for the more well-understood ones.*
>
> (Hewett, 2002, p. 142)

Dave Everitt (an artist) writes about a 'conflict of specialisations' in collaborations:

> *Artists who desire to work with technology are also often faced with a formidable wall of specialism or—particularly in university settings —even protocol, which makes it difficult to communicate with (or even locate) the very technologists who can enable them to realise their ideas. Further, some less visible access needs can also be completely overlooked in environments that profess to be accessible.*
>
> *During many conversations with artists working in this area, common experiences in overcoming 'concept barriers' often crop up. Technologists are often educated to be concrete thinkers who use language precisely. The same phrase can thus mean different things to people with differing backgrounds. The artist attempting to describe a process can end up finding that the technologist interprets their explanation in a more literal manner, one that matches, say, and a computer programming language. Conversely, in order to collaborate, a technologist may require more rigorous logical thinking from the artist in order to understand and then achieve the desired result.*
>
> (Everitt, Turner, Quantrill, & Robson, 2002, p. 10)

McCormack writes about 'ultimate control' of an artwork:

> *inevitably, to really exploit and understand the process I am referring to, you must write your own software. This puts you on more basic terms with the machine, it allows the direct implementation of ideas, as opposed to interpretation through another's constraints*
>
> (McCormack, 1994, p. 23).

John Maeda at one time advocated that artists who make art with computers should learn to program (and wrote a book to help them—(Maeda, 1999)) though he has since revised this opinion (Maeda & Burns, 2004, p. iv)).

On the other hand, many commentators see no problem with artists using technologists as mediators, as a human 'end-user-programming' interface. One justification is to ease the technological complexity of computing: "Due to the technical complexity and the expertise necessary for making such works, it is common for artists to collaborate with technologists in order to realize their ideas" (E. Edmonds et al., 2003, p. 1). The technological complexity of computing is not a key part of the artwork.

End-User Programming

I will argue in Chapter 5 that there is no fundamental ontological distinction between using and programming a computer. This means it is difficult to classify computing systems as being either applications or programming environments. The artist Roman Verostko in Wilson's book (S. Wilson, 2002, p. 317): "Ironically, as the programs get more customisable, the process of setting up options begins to look more and more like programming."

The spreadsheet is the canonical example, and Nardi's work on spreadsheets, amongst other formal languages for end users (Nardi, 1993 Chapter 3) highlights the power and expressivity afforded to end-users who program. For this reason, this section contains a variety of literature, which is not always expressly 'end-user programming' related, but is aimed at making human comprehension of programming easier. I have tried to avoid analysis of specific end-user programming products, except by way of illustration, firstly because of the difficulty of distinguishing them from other applications, secondly because the more general

principles and perspectives are more applicable from a design for creativity support viewpoint. Some specific programming environments are analysed closely in the course of the rest of the thesis.

EDITOR STYLE

Balaban et al. describe computing editors (not just for programs) thus: "The editor is the management system that supports generation, update, modification, testing, running applications, providing feedback, and system integration. As such, modern editors need to provide services that go beyond the immediate command–reaction character of traditional editors. These objectives are achieved by enhancing editors with end-user programming capabilities" (Balaban, Barzilay, & Elhadad, 2002, p. 640).

When considering the interfaces of more complex and flexible applications it is useful to follow Balaban et al.'s classification of editors into extensional and intentional editors. Extensional editors (commonly known as WYSIWYG editors) operate directly on the document-as-artefact, for example in Microsoft Word, and intentional editors operate on the underlying structures of creations) are edited, for example in the LaTeX document editor.

Intentional editors can be seen as examples of programming environments, in that the user specifies abstract operations, which are computed in order to meet the user's goal. Extensional editors abstract the user's edits (to varying degrees, depending on the power of the editor), resulting in a more indirect form of computational specification, but this gives the user little understanding of the computational effect of his actions, being unable to see the underlying structure of the document (grappling with Microsoft Word's (http://office.microsoft.com/) automatic item numbering system with items larger than one paragraph is common case in point.) Balaban et al. claim that the optimum in End-User Programmability lies in providing b*oth intentional and extensional* editing capabilities to the user, saying that a double-view development tool "allows beginners to work in a purely extensional environment and switch to the structured view from time to time for exploring the structure that is being created. In later stages, users learn the meaning of the structure and its usefulness—they casually start editing the structure to

better represent their intended structure. Eventually, they fully exploit the two views, switching as needed" (p. 649). This seems logical, although they did not back up their claim with empirical evidence. Many commercial editors do indeed appear to combine both extensional and intentional views. For example, Corel WordPerfect[9] has long had both views. Although the intentional view of a document is now hidden by default, this feature is particularly valued by authors of highly-structured documents, for example playwrights. Many editors display enough of a document's (usually simple) underlying structure for all of it to be comprehended and directly manipulated via a GUI—Adobe Photoshop[10] is a good example of this, because it shows the many layers which can make up an image, allowing each to be isolated and edited as necessary. Another can be found in music software, such as Cakewalk's products[11], which have intentional views that show the MIDI parameters of, say, all the notes in a track, so the composer can correct any anomalies.

INTUITIONAL EUP VS. EXPOSITIONAL EUP

The EUP literature has two major streams, what I call the *intuitional approach* and the *expositional approach*. There are slight parallels to Balaban et al.'s intentional and extensional editors, in that the computation is more or less hidden in each. The *intuitional approach* requires enough AI capability for the computer to "intuit" the programmatic meaning of a user's instruction, such as by giving examples or advice, or in natural language. The *expositional approach* makes little use of AI, and much use of mechanical interface design to "expose" the mechanics of computation in ways that are easy to understand—through navigation, graphics and simple design.

Let us consider first the intuitional approach to End-User Programming first. A figurehead of the intuitional approach is Henry Lieberman who has, with his colleagues, developed various systems for achieving this in specific domains, including interfaces which give and take advice (Lieberman, 2001a, Lieberman, 1995), and natural language programming based upon commonsense reasoning (most recently (Liu & Lieberman,

9 http://www.corel.com/

10 http://www.adobe.com/products/photoshop/

11 http://www.cakewalk.com/

2005)). In a book entitled "Your Wish is My Command" (Lieberman, 2001b), Lieberman and others (notably Repenning and Perrone, who published a paper on the same topic (Repenning & Perrone, 2000)) discuss the possible uses for programming by example (or by demonstration). The principal idea of programming by example is that if the user demonstrates to the computer several examples of how to carry out a task, the computer should be able to deduce the general principles at work, and thus carry out similar tasks autonomously. In their recent work on programming by natural language, Liu and Lieberman describe Metafor, which uses CyC, a large database of common-sense statements in natural language, from which it is possible for a computer to infer the programmatic meanings of natural language statements (Liu & Lieberman, 2005) (Figure 17).

```
INPUT: "There is a bar with a bartender
who makes drinks."

OUTPUT:def __main__():
    class bar:
        the_bartender = bartender()
    class bartender:
        def make(drink):
    pass
```

Figure 17. An example of Metafor's natural language capability
(Liu & Lieberman, 2005)

For all but the most constrained situations (or numerous examples), however, a high degree of artificial intelligence is necessary for the computer to infer the meanings of variations between examples. This problem makes the authors' stated beliefs that "[programming by example] will some day make it possible for interfaces to effectively say to the user, 'Your wish is my command!'" a long way from being realised. And even if the dream is realised, having a computer realise your every desire is an odd form of creativity support: How would the computer respond to a wish for a new piece of music? How would a user be able to modify the computational system, the AI, which generates such a tool? More generally, a common problem in software development today is that often, stakeholders do not know how to describe what software they

want. In those situations, it takes more intelligence, creativity, and iteration to establish what software a stakeholder wants, than it does to then make the desired software. Even assuming AI develops to the point where it can be creative and intelligent enough to provide such exploratory software, it would simply be a cheaper, quicker equivalent to the expert programming that humans provide today, still without providing understanding and control of the computing medium itself to the stakeholder. I consider intuitional paradigms to be simulacra of the role that technologists play in collaborations today (of which we shall discover more in Chapters 6 and 7).

The expositional approach is characterised by an (eventual) exposition of the mechanics of computing; a direct representation in the interface of what is actually taking place within a computer. Most conventional programming languages are expositional. Of those that are specifically for end users, visual programming languages are the most accessible and Smalltalk is perhaps the most powerful (for a full explanation and description of why, see Chapter 5; for an heuristic evaluation of Squeak Smalltalk, see Appendix 2 (on CD)).

From the point of view of creativity support, it seems to me that being able to perceive what is actually happening during programming, rather than AI-derived abstractions of it, would allow creative users to understand and control the medium to a finer degree. However, an idealised intuitional approach requires no special knowledge—we would be able to program by using the linguistic skills we already possess, whereas an idealised expositional approach would need to provide a way for us to gain the knowledge required to understand the exposition of the mechanics of computing. One way of doing this is to find ways to culture people in such knowledge, in the same way as people are cultured, or educated, in maths, music and literature. Another approach, hardly in conflict, is to design technology and interfaces which manage a gradual transition, a smooth learning curve, from what we already know to what we need to know - from novice to expert, from consumer to designer. The next two sections deal with these aspects.

DEVELOPING A PROGRAMMING CULTURE

Simply developing the technology for end-user programming is unlikely to be sufficient to ensure its adoption. MacLean et al. (MacLean, Carter, Lövstrand, & Moran, 1990), in their introduction to the "Buttons" end-user programming paradigm, emphasise the importance of developing a culture around tailoring, as well as the technology. Specifically, they state,

> *if workers have no expectation of controlling changes to the system, they are not in a good position to understand what changes might be possible, let alone [to] make them happen themselves. … Our goal is to give the worker a feeling of ownership of the system, to feel in control of changing the system and to understand what can be changed.*
>
> (p. 176)

This goal is shared with other Participatory Design approaches (e.g. that of Ehn and Kyng (1988)). They introduced a "handyman", a member of the Buttons design team, to the workers, to act as a catalyst for the development of a culture in which workers expected to be able to control changes. The handyman bridges between 'workers' and 'programmers', by being able to respond to workers' immediate needs, as well as by communicating user needs to programmers for longer-term development. In the context of their research, the handyman also observed and interviewed participants about how the culture of use of Buttons evolved. Two striking changes were that users moved from talking about Buttons in impersonal terms to being their buttons, as they became more customised. Secondly, a kind of social system grew, where people of all levels of expertise shared their modifications with others via email.

Procedural literacy is the treatment of deep programming education as not a specialised discipline learned by engineers, but as a fundamental language of self-expression used by anyone. Two proponents of procedural literacy are Ken Perlin and Mary Flanagan, who are collaborating on the RAPUNSEL (Realtime Applied Programming for Underrepresented Students' Early Literacy) project. The project's manifesto states:

> *We want to improve learning by contextualising concepts and problem solving inside structures which will give a base for making abstract problems "real." We want to facilitate learning through play systems which invoke peer to peer interaction, sharing, and instant feedback. We are working to change the perception of role and place of science, math, and technology in order to promote associated fields of work. Most of all, we are working to increase comfort level with technology by making programming a part of everyday life.*
>
> (http://www.maryflanagan.com/rapunsel/manifesto.htm)

The RAPUNSEL project focuses on learning for middle-school girls, by using programming concepts in a multiplayer game that "breaks some social and technical stereotypes for both girls and boys"[12] . The software has several working prototypes, which Perlin and Flanagan demonstrated at ISEA'04 (Flanagan & Perlin, 2004), but few discernable publications. Ken Kahn developed the ToonTalk system with a similar aim (Kahn, 1996).

From a new media point creation of view, Mateas has this to offer:

> *Procedurally illiterate new media practitioners are confined to producing those interactive systems that happen to be easy to produce within existing authoring tools. Even those practitioners who don't themselves write much code will find themselves on interdisciplinary collaborative teams of artists, designers and programmers. Such collaborations are often doomed to failure because of the inability to communicate across the cultural divide between the artists and programmers.*
>
> (Mateas, 2005)

On the other hand, anti-procedural literacists say that although programming provides a useful context in which to learn, say, mathematics and physics, there is no evidence that learning to program affords higher-level cognitive skills development nor transfer (Palumbo, 1990).

---

[12] http://www.maryflanagan.com/rapunsel/research.htm

As Alan Kay (Kay & Goldberg, 1977) (inventor of the term "Personal Computing" (Kay, 1993)) and Seymour Papert (designer of the Logo programming language) (Papert, 1980) claim, the computer offers unique possibilities for knowledge transfer and problem-solving, particularly in an educational context. Kay uses the concept of a 'meta-medium for creative thought', embodied by the programming language Smalltalk, to illustrate his claim. Papert advocates "teaching the computer" through programming as a path for people to improve problem-solving skills. The use of programming languages as means to empower individuals to investigate and solve problems is shared by Mary Flanagan and Ken Perlin, an artist and computer scientist who are collaborating to make a graphical programming language for children (Flanagan & Perlin, 2004).

There are aesthetic as well as intellectual benefits to software engineering, writes Miller Puckette, inventor of the Max/MSP system:

> *Well-designed software enhances the workspace in the same way that well-designed furniture does, not only in functionality but also in the stylistic choices that enhance, and do not depress, the quality of our environment.*
>
> (Puckette, 2002)

For a brief heuristic evaluation of Max/MSP, see Appendix 2 (on CD). Max/MSP is also described in Chapter 5 (pp. 164-166).

METADESIGN: DESIGNER–CONSUMER AND EXPERT–NOVICE TRANSITIONS

Metadesign (Fischer et al., 2004; Giaccardi & Fischer, 2005) is a methodology for creating new media and software environments that allow the owners of problems to act as designers throughout the lifespan of the system—design for designers, as it were. Fischer describes the following requirements for meta-design, supported by empirical evidence (Fischer, 2002):

- Software systems must evolve; they cannot be completely designed prior to use
- Software systems must evolve at the hands of users

- Software systems must be designed for evolution

Metadesign extends the traditional notion of system design beyond the original development of a system to include an ongoing process in which the stakeholders of the system become co-developers. For example, meta-design concepts embedded in Microsoft Word include: (1) users can tailor the system by setting different parameters as their personal preferences; (2) they can not only use spelling correctors, but they can extend the spelling dictionaries; (3) they can write macros to create new operations; and, (4) they can create programs in VisualBasic to extend the functionality of the system" (Fischer, 2002).

In Glaser's review of Expert/novice research (R. Glaser & Chi, 1988), he suggests that novices' representations of a problem are organised around the literal objects and events in the problem. Experts' knowledge, on the other hand, is based on more abstract principles that subsume the literality of the specific problem:

> *For example, in our studies with mechanics problems, novices classify problems on a surface level, according to the physical properties of a situation—a spring problem or an inclined plane problem. Experts categorize problems at a higher level, in terms of applicable physics principles—a Newton's second law problem, a conservation of energy problem.*
>
> (R. Glaser & Chi, 1988)

Applied to the programming domain, this research implies that a system that is able to show the abstractions that guide a literal concept is more likely to help a novice transition into an expert. For an expert in another field (namely art), the ability to express expert knowledge in a way that is comprehensible to the "novice" computer is also desirable.

Fischer goes on to discuss open source software as an example of collaborative design, and the role of the open source movement as meta-design. He identified the following necessary principles:

- making changes must seem possible
- changes must be technically feasible
- benefits must be perceived

- open source environments must support tasks that people engage in
- low barriers must exist to sharing changes

Fischer (2002) characterises the consumer-designer spectrum in terms of roles of both people and media as follows:

Consumer End
passive consumer
active consumer
power users
local developers
domain designer
meta-designer
Designer End

TV is an example of a "passive consumer" medium; current computer software is an example of the "active consumer" role. Fischer envisions that computational media should move towards the 'metadesign' end of the spectrum.

BUTTONS

MacLean et al. (1990) propose a system called 'buttons' to smooth their tailorability mountain, which are screen objects in the Xerox Lisp environment which carry out an action when pressed. Buttons can exist in the environment's desktop, or within individual documents, and can be emailed from one user to another. There is a range of techniques for tailoring these buttons. Firstly, the buttons can be arranged on screen, allowing users to group them appropriately to a given work context. Secondly, buttons can be duplicated, allowing copies to be specialised to slightly different tasks. Thirdly, certain parameters of each button can be modified, allowing users to 'tweak' the action to an appropriate context. Fourthly, the Lisp code behind the button can be edited in the same way as other program code. MacLean et al. describe 'tinkering' as code modification by a non-expert in Lisp, in addition to programming by experienced Lisp programmers.

ASPECT-ORIENTED AND TABLE-ORIENTED PROGRAMMING

Designers of underlying technology are also beginning to pay attention to end-user programming, and to supporting the increasing capabilities of programmers. "Programmers using Java or .Net today can write programs that only the best could write 10 or 15 years ago. People are really struggling for the next step after object-oriented programming"—Gregor Kiczales in Ricadela (2002). Aspect-Oriented Programming (AOP) is beginning to emerge as a follow-on to Object-Oriented Programming (Elrad, Filman, & Bader, 2001). Pioneered by Gregor Kiczales during more than a decade of international research (Ricadela, 2002), AOP is based on the premise that "many requirements do not decompose neatly into behaviour centred on a single locus"(Elrad et al., 2001), the program is specified by "separately specifying the various concerns (properties or areas of interest) of a system and some description of their relationships, and then relying on mechanisms in the underlying AOP environment to weave or compose them together." For example, a user may wish specify that he wants to see his bank statement on screen, whilst a network administrator may wish to specify that all transactions be secure. These separate requirements are intertwined in a way that satisfies both. Elrad writes "AOP systems offer implicit invocation mechanisms for invoking behaviour in code whose writers were unaware of the additional concerns."

Somewhat related to AOP is table-oriented programming (TOP), which is simpler. Similar items are grouped among a single dimension of a table, allowing many types of grouping (whereas in OOP, for example, items are usually only grouped by class). Information visualisation expert's Edward Tufte's website serves as a forum for practitioners to discuss information visualisation. In a topic about programming as an information design problem, Jeffrey Berg offers this comment on both AOP and TOP:

> *Either way, the current shift in ideas about programming theory from pure OOP is that the information shown by the programming method is by default required to convey the program's structure and ability. This information should be clear, concise, and easily changed by programmers who may end up looking at the code in the future.*
>
> (Berg, 2003)

AOP and TOP offer potential to smooth the transition from consumer to designer by separating the concerns of various stakeholders of a program. This means that anyone with a particular concern need not be overwhelmed or distracted by other concerns. On the other hand, many AOP/TOP researchers envision that deciding and laying out the aspects of a program would be a technical discipline similar in specialisation to programming today, which would mean that fundamental system design would remain beyond the reach of end-users, who would be limited to the boundaries of the aspect which had been decided for them.

VISUAL PROGRAMMING ENVIRONMENTS (VPES)

Myers (Myers, 1990) characterised Visual Programming as a way to "specify a program in a two-(or more)-dimensional fashion", hinting at the uses of graphics and positioning to highlight the structure of the program and reduce the cognitive load on the programmer. Myers' definition has the caveat that most text-centric programming environments actually also rely on 2D layout (line breaks and indentation) to impart many of the same qualities, but the indicated structure here is generally w*ithin* program units, rather than b*etween* them.

Costagliola et al. (Costagliola, Delucia, Orefece, & Polese, 2002) characterised VPEs according to graphical attributes (lexicon), syntactic attributes and semantic attributes, and found two main classes: geometrically important languages, where the syntax and semantics are conveyed simply by the position of elements, and connection-based languages, which convey these things by depicting connections between elements.

Lieberman (Lieberman, 2001a) writes "Often the visual information serves as advice to the reader in how to interpret textual information, or vice-versa" which can be seen as a corroboration of Balaban et al.'s sug-

gestion that varieties of editors should be combined (in this case including visual editors).

Fry (Fry, 1997) cites non-scalability as a limitation of graphical languages. Words are more concise at large scales, and the programmer is confronted with the task of visually laying out the program for comprehensibility as well as creating the structure of programming.

CODE TYPOGRAPHY AND LITERATE PROGRAMMING

Donald Knuth introduced the idea of literate programming (Knuth, 1984), that the software and documentation should be as one, by writing your documentation in amongst your program, with the idea that a printout can be produced which contains both the code and a comprehensible rationale for it.Ron Baecker et al. (Baecker, DiGiano, & Marcus, 1997, Baecker & Marcus, 1990) extended Literate Programming by experimenting with typographic styles for code, employing graphic design principles to best represent its structure and meaning. Literate programming is geared towards printed documentation—treating the code as a technical publication complete with tables of contents, footnotes and cross-references. Whilst this is a highly-navigable and readable (for code) document, the problem with print is that for all but rigidly-finished programs, any printed code very rapidly becomes obsolete. However, this work was created in the light of the low-resolution monochrome displays available at the time, and is due an on-screen, interactive update. Most common code editors today pay homage to Baecker et al in that they display different language elements in different colours, and some in bold or italics. Such elements are in one sense redundant because the compiler ignores them, but, to the human party in the communication, such redundancy is a highly useful part of the language, and should not be dismissed lightly.

DEBUGGING/DYNAMIC VISUALISATION

> *Debugging is the second oldest profession in software engineering.*
>
> Thomas G. Moher (Moher, 1988)

Henry Lieberman, in his introduction to a special journal issue on "The Debugging Scandal", writes,

> *What borders on scandal is the fact that the computer science community as a whole has largely ignored the debugging problem. This is inexcusable, considering the vast economic cost of debugging and emotional toll buggy software takes on users and programmers. Today's commercial programming environments provide debugging tools that are little better than the tools that came with programming environments thirty years ago. It is a sad commentary on the state of the art that many programmers name "inserting print statements" as their debugging technique of choice.*
>
> (Lieberman, 1997)

Eisenstadt (Eisenstadt, 1997) conducted a phenomenological study of the debugging process by gathering self-reports from programmers of particularly intractable bugs, and categorised the results according to the following (listed in order of frequency of occurrence in 51 reports):

Dimension 1: Why a bug was hard to find. Often the symptom was far-removed (in space and/or time) from the cause (is the level of removal a programmer is able to detect related to his experience?). Sometimes switching on the debugging tools made the bug go away (nicknamed the "Heisenbug uncertainty principle"). Sometimes the programmer misread the program, or misunderstood the programming environment. Sometimes the programmer was struggling through spaghetti code.

Dimension 2: How the bug was found. Most often a programmer gathers data by planting print statements or breakpoints in the code. The most advanced such technique is to log the behaviour of the program in two different execution runs, and compare using a "diff" utility. The next most common category broadly describes non-testing activities—either hand-simulation, background reading or thinking about something else for a while. The third most common technique is having an expert (or other programmer) recognise a cliché.

Dimension 3: The root cause of the bug. The most common root cause of stubborn bugs is memory management – running out of or overwriting memory. The next most common is faults in third-party hardware or software. The solutions here are either to develop workarounds or wait for the vendor to provide corrective measures. The third most

common cause is faulty design logic – an unanticipated situation had occurred. The fourth is wrong initialisation of variables, types or scopes. The fifth is wrong usage of variable names. The sixth is a lexical mistake such as a typo or ambiguity in the syntax. The seventh most common category is bugs that remain unsolved. The eighth category is a language misunderstanding by the programmer (perhaps skewed because of the self-report nature of the survey), and the ninth is the result of a subtle behaviour by the user or programmer, for example pressing several keys at once.

Eisenstadt uses these results to suggest several ways forward, referring to other papers in the same issue:

- Computable bug-checks should be computed on request, rather than be deduced by the user. For example, memory-leak detection and type checking.

- Displayable states should be displayed on request, rather than be deduced by the user. Minimising deductive work is an important aspect of tools such as Zstep95 (Lieberman & Fry, 1997)

- Atomic user-goals should be mapped onto atomic actions. In other words, try to anticipate some of the programmer's likely intentions (e.g. Fry, 1997)

- Allow full functionality at all times. Debugging environments which prevent access to certain facilities just make matters worse. The Kansas/Oz environment described in the accompanying paper by Smith et al. (1997) pushes this notion to its logical limits.

- Viewers should be provided for "key players" (any evaluable expression) rather than just "variables". Several of the papers in this volume, particularly that of Baecker et al., take to heart the notion that more than variable-watching is at issue during the debugging process!

- Provide a variety of navigation tools at different levels of granularity. Changing granularity is one of the hallmarks of the system underlying the work of Domingue and Mulholland (Domingue

& Mulholland, 1997) in this volume, and offers programmers the opportunity to see appropriate views at appropriate times.

Dynamic program visualisation (visualisation of the execution of the program) is particularly useful when debugging. Baecker et al. (Baecker, DiGiano, & Marcus, 1997) present two approaches to dynamic program visualisation: algorithm animation and interactive auralisation. Algorithm animation is useful for abstract understanding and comparison of similar algorithms, but not so much use for generalised debugging beyond the visual presentation of values that could straightforwardly be automatically generated. Currently such animations need to be carefully crafted for a specific algorithm or group of algorithms (the classic example is comparing sort algorithms).

Baecker et al.'s version of auralisation of programs allows the programmer to insert audio probes into the program, which trigger a particular sound in response to a particular event. The result is equivalent to listening to a car engine with an iron bar, and makes it easier to gain an intuitive understanding of the sound of programs and how it might differ when a bug is encountered. However, the sound may distract if the program has another audio component.

A number of implications for the design of debugging technology can be derived from Baecker et al.'s work, as they write:

- •*The need for graphic design expertise in developing visualisation technology;*
- *The importance of variety, cleanliness, and simplicity in the choice of visual representations;*
- *The desirability of using sounds as well as images;*
- *The importance of speed controls for program playback;*
- *The desirability of being able to attach debugging probes unobtrusively to source code; and*
- *The need for tools for enhanced static display of source code and for dynamic display of program execution.*

Summary

I have covered a wide range of topics in this review, which will inform argument and analysis in my new work. A summary of these topics follows.

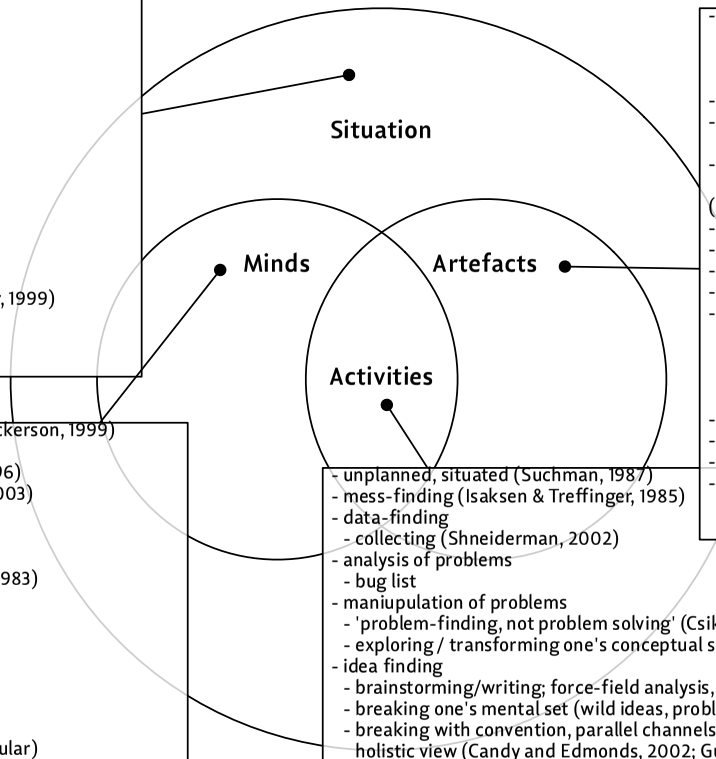
Creativity has many different definitions and qualities. A typical definition is "the ability to produce work that is both novel and appropriate". For this research we are particularly interested in Margaret Boden's notion of *conceptual spaces* (e.g. Boden, 1994). Conceptual spaces can be explored or transformed in order to produce radically original ideas. As exploration takes place, the constraints of the space become apparent. It follows that if constraints can be transformed when needed, those constraints support creativity at those boundaries. It also follows that if constraints cannot be transformed when needed, creativity is limited by these boundaries.

There are many factors that can help to support creativity. I have arranged these into qualities of situations, minds, artefacts and bodies, though there are overlaps between these. Qualities of situations include "peaceful setting", "adequate resources", and "synergistic extrinsic motivation". Qualities of minds include "intrinsic motivation", "creativity-relevant skills", "collaboration skills" (at conflict with "independence"), "self-confidence" and "synaesthesia". Qualities of artefacts include "simplicity", "externalisations of knowledge" and "stimulation of emotion". Qualities of activities include "problem-finding", "problem-solving", "collaboration" and "iteration". The literature on creativity support has been placed on the Venn diagram introduced in Chapter 1, and is presented overleaf.

What do interactive artists want to do with computers? Anything! Interactive art stems from a rich history of involving the audience in art. It represents some of the broadest and most demanding uses of computers, due to the unusual juxtapositions of hardware and software, the realtime response it often requires, and the variety of forms it can take. Opinions

vary on whether or not interactive art must be programmed by the artist. A moderate view, adopted in this thesis, is that artist need not program if the artwork does not involve computing p*er se.*

Metadesign is the concept of designing (tools) for designers. End-User Programming (EUP) is a type of metadesign, and comprises many approaches. Perhaps the largest approaches are technological (making programming easier) and pedagogical (widening the culture of programming), though many projects combine both. Of the technological approaches, I distinguish between intuitional and expositional paradigms. Intuitional paradigms use AI to 'intuit' the programmer's intent. Expositional paradigms gradually 'expose' the workings of computing in an easy-to-manipulate interface. I argue that intuitional paradigms could inhibit creativity because they override subjective interpretation of the programmer's intent and deny direct engagement with the underlying computing. Of the expositional EUP approaches, 'Buttons', aspect-oriented programming and visual programming were described. Debugging/Dynamic program visualisation was identified as an understudied topic.



- peaceful setting / immersion (Couger, 1996; Candy & Edmonds, 2002)
- synergistic extrinsic motivation (Amabile, 1987)
- supportive cultural atmosphere
  - ethnic (e.g. Lubert, 1990)
  - institutional (Williams & Yang, 1999)
    - beauracracy (processes and people)
    - stability
    - freedom / providing opportunities (Nickerson, 1999)
    - good project management
    - sufficient resources
    - encouragement, recognition and reward (Amabile, 1987, Nickerson, 1999)
    - collaborative/cooperative atmosphere
      - diversity (Fischer, 2005)
    - relaxed division of labour / relaxed hierarchies
    - teaching creativity techniques (Nickerson, 1999; Couger, 1999)
- sufficient time
- sense of challenge
- balance (not too much of any of these) (Nickerson, 1999)

- externalisations of knowledge systems / cognition
  - boundary objects (Star, 1989; Fischer, 2002)
    - bridging spatial, temporal, conceptual and technological boundaries
  - design journal (Guindon & Curtis, 1988)
- reusable schemas / patterns
- simulation tools
  - displays of constraints
- smooth transition from design to implementation
  - representation of abstract design

(Resnick et al., 2005):
- low threshold, high ceiling, wide walls
- supports open interchange
- as simple as possible
- carefully-chosen black boxes
- stimulation of emotional states
  - engagement/personal meaning (Fischer, 2005; Nakakoji, 2005)
  - embodiment (Nakakoji, 2005)
  - trust (Nakakoji, 2005)
  - artefact inventor enjoys using
- user suggestions balanced with observation
- has been metadesigned, allows acquisition of ownership (Fischer, 2004)
- evaluate the tools
- three roles (Nakakoji, 2005)
  - dumbbells (supporting skills development)
  - running shoes and skis (aiding new experience)

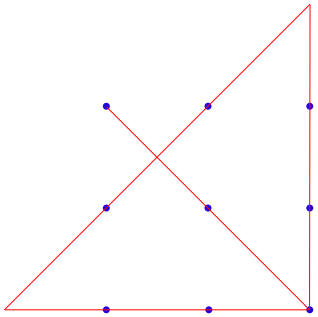
- intrinsic motivation / intention, purpose (Amabile, 1987; Nickerson, 1999)
- focus on mastery (Nickerson, 1999)
- insight (Weisberg & Alba, 1981; Baker-Sennett and Ceci, 1996)
- synaesthesia (Ramachandran & Hubbard, 2001a, 2001b, 2003)

(Barron & Harrington, 1974):
- independence
- self-confidence
- curiosity, attraction to / coping with complexity (Amabile, 1983)
- aesthetic orientation
- risk-taking

(Lubart, 1990; Lubart & Sternberg, 1995)
- self-efficacy
- tolerate ambiguity

(Feist, 1999)
- relatively high introversion
- anxious, emotionally adaptable, impulsive (artists in particular)
- emotionally stable, venturesome and self-assured (scientists in particular)
- nonconformity
- domain-relevant knowledge (Amabile, 1983)
- creativity-relevant skills (Amabile, 1983; Amabile, 2001)
  - knowledge of heuristics for generating novel ideas (Couger, 1996)
    - divergent/convergent thinking (Guilford, 1950)
  - high energy
  - concentrated effort
  - ability to set aside problems
- collaborative skill (Amabile, 2001)
  - experience
  - compatibility of problem-solving styles
  - leader's skill in managing collaboration
  - clear understanding of roles and responsibilities
- self-management / metacognitive skills (Nickerson, 1999)

- unplanned, situated (Suchman, 1987)
- mess-finding (Isaksen & Treffinger, 1985)
- data-finding
  - collecting (Shneiderman, 2002)
- analysis of problems
  - bug list
- maniupulation of problems
  - 'problem-finding, not problem solving' (Csikszentmihalyi, 1999)
  - exploring / transforming one's conceptual space (Boden, 2004)
- idea finding
  - brainstorming/writing; force-field analysis, and similar.
  - breaking one's mental set (wild ideas, problem reversal)
  - breaking with convention, parallel channels of exploration, holistic view (Candy and Edmonds, 2002; Guindon & Harris, 1988)
- solution finding
  - near-term experimentation (Terry & Mynatt)
  - relating ideas to each other (Shneiderman, 2002)
- acceptance finding
  - contributing knowledge to the field (e.g. Shneiderman, 2002)
- reflection-in-action (Schoen, 1983)
- iteration (e.g. Ullman et al. 1988; Isaksen & Treffinger, 1985)
- Searching, Visualising, Consulting, Thinking, Exploring, Composing, Reviewing, Disseminating (Shneiderman, 2002)
- Collaboration
  - forming reflective communities, communities of interest (Fischer, 2001, 2003, 2005)
  - high interaction (Amabile, 2001)
  - high information-sharing (Amabile, 2001)
  - strong coordination processes
  - mechanisms for developing shared understanding

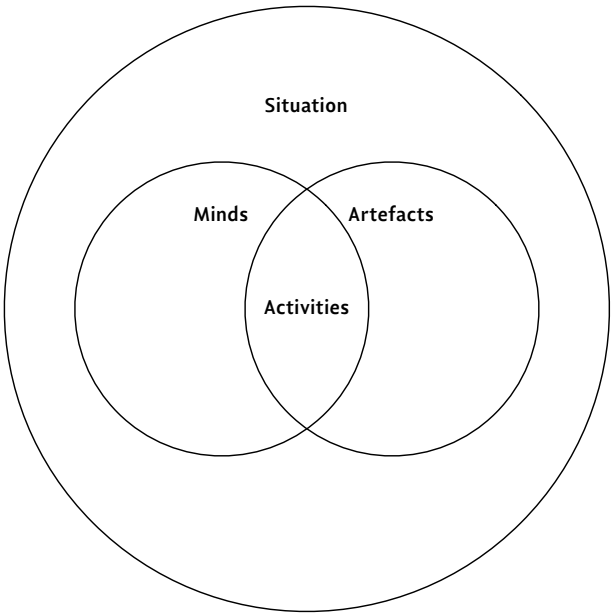


3. Methodology

Introduction

In this chapter, I describe and justify the methodological approach taken in this thesis. To avoid confusion between the research methodology discussed in this chapter, and the interactive art-making methodology under study, I will term the art-making methodology 'art-methodology' during this chapter. The literature review has focussed our attention on the intersection of the problem domains of creativity support and of interactive art practice. Let us also remind ourselves of the research question, and the accompanying Venn diagram:

Q: **In a situation which otherwise supports creativity, what would be useful technology and related [art-]methodology to help people to make interactive art?**



I have specified the desired situation to be "actual interactive art-making in a collaboration", and the people to be experienced artists and programmers who are collaborating, each, by virtue of their experience, tending to possess the qualities identified in the literature review to be supportive of creative collaboration.

This leaves us with many methodological questions, firstly of how to identify needs for tools (examples of 'artefacts' in the Venn diagram) which not been met, and secondly of how to identify needs for art-methodologies (examples of 'activities' in the diagram), remembering that we cannot consider tools outside of the context in which they are used—if we were to concentrate only on 'improving' tools, without considering the context in which those tools are used, we run the risk of ignoring important, and possibly negative implications that those tools have. Once we have identified those needs, we need to identify and/or design tools and art-methodologies that would satisfy those needs.

The NSF workshop on Creativity Support Tools included a breakout session on the evaluation of creativity support tools. The group found they shared strong belief in one principle, that "there is a *family* of evaluation techniques which must be brought to bear in order to *converge* on the key issues involved" (Shneiderman, 1980, p. 14). This principle suggests employing a variety of techniques in order to come to the most complete, least flawed understanding of the issues, and is exemplified by the variety of techniques that I will employ and brought together.

Briefly, my approach has been as follows. The first thing to do is to explore tools—to become well-acquainted with existing tools for making interactive art, and their creativity support features. This process is similar to a critical literature review, but with the inclusion of an argument, based upon that literature, for certain needs of technology. The second thing to do is to explore art-methodology. Since this is an exploration of a social process, without imposing pre-defined goals, we need to use a research method that will provide us with the art-methodological needs. Grounded theory analysis will satisfy our requirements (on the condition that there *are* art-methodological needs). Thirdly, we can use the tools expertise gained in the first stage with the art-methodology expertise gained in the second to recommend useful tools and related art-methodologies. The recommendations are, in part, a software engineering exercise, and in part, an implementation of the useful art-methodologies already identified, which takes into account the new tools. Finally, to evaluate the tools and art-methodologies, they need to be used in the context of actual art collaborations. I participated in one such collaboration, involving prototypes of the recommended technologies and art-methodologies. I observed the collaboration and interviewed the artist to understand the improvements and opportunities for further research.

To answer these methodological questions, I have drawn from Michael Crotty's book 'The Foundations of Social Research' (Crotty, 1998) and the in-person guidance of Linda Candy, an expert in research methodologies for studying creativity, to form my methodology. Both sensibly advocate deciding high-level approaches to a problem before choosing specific methods and tools. With a more specific and precise problem

definition it might be a simpler matter to choose a set of methods, perhaps because similar problems exist with a history of successful methodologies. Tackling the problem from a high level downwards, starting at the epistemological level, invites researchers to give careful consideration to taxonomy of methodological approaches. This systematic consideration allows researchers to apply a variety of appropriate methods and sources of information to increase rigour, and to harness that variety to triangulate solutions.

So to present the approach in detail in this chapter, I begin with my epistemological and ontological stances, which describes the type of knowledge that this research deals with. I go on to characterise the type of creative situation and people under study in order to relate this research to other creativity research and to further specify the *situation* and *people* in the Venn diagram. Then I discuss the strategic approach to deciding the goals of each stage of the research and what methods to use at each stage. Finally, I describe the specific methods and tools I use.

Relativism, Realism and Constructionism

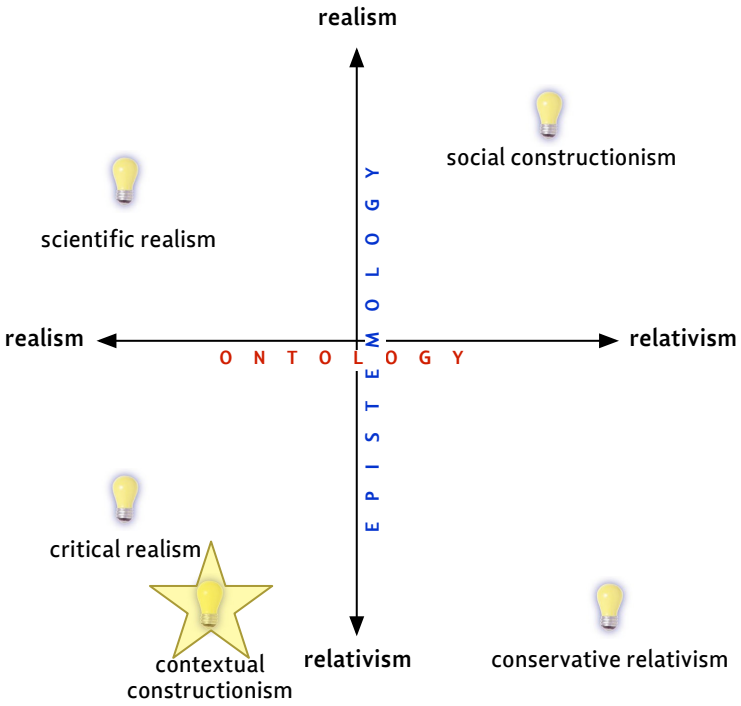
Epistemology deals with the nature of knowledge, or what it is possible to *know* about reality. Ontology deals with what exists in reality for us to know *about*. Declaring a stance on what exists and what it is possible to know will inform the meaning and validity of all which follows. I have adopted a *contextual constructionist stance,* as described by Crotty (Crotty, 1998), and have adapted his description for this section, and have refined it with Michael Larkin's concise summary of *relativism* and *realism* (Larkin, 2004) and by reading the cited sources as suggested by both (particularly Cromby and Nightingale (1999)). Different researchers and philosophers impose different interpretations and arrangements of the terms used; these are mine, based upon my understanding of Crotty's and Larkin's.

Realism holds that reality is external to the mind and is independent of our representations of it. On the other hand, *relativism* holds that any such external world is inaccessible to us, since it is only our relation to it that creates meaning. Views on epistemology and ontology can be relativist or realist. Larkin arranges these into four broad combinations:

- *Relativist ontology plus Relativist epistemology (e.g. conservative relativism)*
- *Relativist ontology plus Realist epistemology (e.g. social constructionism)*
- *Realist ontology plus Relativist epistemology (e.g. critical realism; contextual constructionism)*
- *Realist ontology plus Realist epistemology (e.g. scientific realism)*

adapted from (Larkin, 2004)

Within these combinations of realism and relativism, there are many possible stances, and to catalogue them all would be beside the point, which is to choose a combination of relativism and realism that comfortably fits the research question. To that end, I present examples of each combination (more examples are in Larkin (2004)) and then critique them in order to focus on a combination to use. The figure below shows my arrangement of epistemology and ontology, ranging from relativism to realism, with the notional examples positioned. The approach I use, *contextual constructionism,* is highlighted with a star.



Conservative relativism (an example of Relativist ontology plus Relativist epistemology) is the doctrine that there is no knowable material reality, and no reliable form of knowledge about it either. Yet,

> *to declare that no utterance can be true because it is a product of the one who utters it, is devastating for the statement itself […] Nobody would take this form of relativism seriously.*
>
> (Bem & de Jong, 1997, p. 70) cited in Larkin (2004).

Social constructionism (an example of Relativist ontology plus Realist epistemology) (Holstein & Miller, 1993; Loseke, 1999; Potter, 1996) also holds that humans cannot access material reality, but that we can access our discourse and subjectivity. Discourse and subjectivity construct our understanding of material reality.

Critical realism and **contextual constructionism** (two examples of Realist ontology plus Relativist epistemology) (Nightingale & Crombie, 1999) hold that there is a material reality which precedes our experiences of it, but that we can only access the entwined relationship between that reality and our perceptions of it. Our reality shapes our discourse and is shaped by it. For contextual constructionism, furthermore, all knowledge is local, provisional and context dependent.

Scientific realism (an example of Realist ontology plus Realist epistemology) (Potter, 1996) assumes that there is a material reality, which precedes our experiences of it. Language is one mode of engagement with this reality, and we can use it to describe and explain reality. Scientific realism holds that science can be fallible, and 'truth' is open to question, but science is seen as a self-correcting model for developing best-fit descriptions of the world.

Let us examine the appropriateness of these approaches to the research question. Both *relativism* and *realism* have their dilemmas and blind spots.

> *While realists shoot themselves in the foot as soon as they represent, relativists do so as soon as they argue. To argue for something is to care, to be positioned, which is immediately non-relativist.*
>
> (Nightingale & Crombie, 1999, p. 6)

These dilemmas and blind spots cause r*elativism* and r*ealism* to often be tempered with each other, for moral, political or pragmatic purposes as much as any particular certainty in an overarching epistemology or ontology.

Classical computer science has adopted a broadly s*cientific realist* approach, perhaps because a computer is a well-defined, well-behaved system in its own right. However, as presaged by Dijkstra's 'software crisis' (Dijkstra, c1997), in which he notes that the presence of human 'interrupts' in computer systems means that software behaviour is no longer so easy to test, we are often compelled to take into account human social perceptions, rather than considering only an objective, 'essential' reality, when studying interactive systems. Despite this importance of human perception, the early days of HCI were still dominated by scientific realist approaches, and some studies still adopt this stance. Shneiderman's early experiments are perhaps the best-known examples of the scientific method in HCI. To take a specific example, Shneiderman et al. (1977) report that experiments on the utility of *detailed* (as distinct to simplified) flowcharts showed no statistical difference between the flowchart and non-flowchart groups, "thereby calling into question the utility of detailed flowcharting". More recently, Good (1999) compared two styles of visual programming languages, and used multiple-choice questions to test for 'program comprehension', saying that data flow languages resulted in 'abstract, functional accounts' while control flow languages resulted in 'low-level, procedural descriptions'. The result itself is clear, but seen in a wider context, it is not clear, for example, which language, if either, would be more useful for any given situation. More generally, scientific realist approaches are most reliable for reaching specific and precise conclusions about specific and precise situations.

So the importance of humans in interactive systems and in creativity means that realist approaches such as scientific realism might not be the most appropriate stances to adopt to answer my research question. Leaving aside scientific realism and the other realist views, then, and also its opposing family of entirely relativist views, in which we cannot ultimately show anything about anything (as I mentioned in its description), we are left with the realist plus relativist *constructionist* combinations, in which knowledge is constructed, either from our representa-

tions of an underlying reality, or from the interplay between reality itself and our representations of it.

Social constructionism, in which our knowledge is constructed *only* by our subjectivity and through social discourse, because that is all we can directly access, has a weakness when applied to this problem because, by its own definition, discourse cannot convey knowledge of subjectivity itself, only representations of it, and so discourse is still prone to subjective perception.

Contextual constructionism overcomes this weakness because it presumes an underlying reality, allowing us to suppose a context shared between people, a shared framework of meaning, such as 'culture', with which individuals are able to approximate each others' worlds, and so are able to approximate a shared understanding. For shared understanding to be the case, however, we must further suppose that this shared understanding is either of an independent reality (in other words, something like scientific realism, already ruled out for this work) or arises from presumed commonalities in our constructed knowledge of that independent reality. The latter stance is known as contextual constructionism. *Contextual constructionism* includes the shared understanding that is, according to Fischer (2001) so important in collaborations like those I am studying, and which is what permits researchers to be able to understand people under study. Because of this, I shall adopt a *contextual constructionist* stance for my research, and shall it explore its further implications here:

Contextual constructionism has its roots in phenomenology, particularly that associated with Edmund Husserl and Alfred Schutz (Loseke, 1999, p. 197), though Heidegger, in his magnum opus *Being and Time* (Heidegger, 1962), transformed Husserl's phenomenology from an epistemological question ("How can we know about the world?") to an ontological question ("How does the world reveal itself to us through our encounters with it?") (Dourish, 2001), which is more appropriate to contextual constructionism.

Schutz, writes Dourish (pp.110, 131-2), extended phenomenology from the individual world to the social world and its intersubjectivity. Phenomenologists, who combine realism and relativism and hold that

subject-object are not separate, must, then, always keep subject-object together. On this matter, Michael Crotty warns:

> *Bringing objectivity and subjectivity together and holding them together throughout the process is hardly characteristic of qualitative research today. Instead a rampant subjectivism seems to be abroad. It can be detected in the turning of phenomenology from a study of phenomena as the immediate objects of experience into a study of experiencing individuals. It is equally detectable in the move taking place in some quarters today to supplant ethnography with an 'autoethnography'.*
>
> (Crotty, 1998, p. 48)

Crotty is warning against the 'embedded' approach of investigators to report their own perceptions as being *just as valid* as a qualitative study of others. To an extent this effect is unavoidable since constructionists hold that meaning is inseparable from perception, but on the other hand, contextual constructionists share meaning with those in a shared context. Crotty points out that humans must make sense of the world from a genuinely historical and social perspective, and it is unwise to ignore those effects, the effects of culture in other words, on perceived meanings (p. 54). Since views of even creativity vary between cultures (e.g. Lubart (1999)), it is likely that I or anyone else would arrive at a different conclusion if my human subjects or I were 18th-Century Japanese, for example. This means that I should declare that my research takes place within my culture as, broadly, a British male with a computer science background residing in urban Australia in the early 21st Century, and reflects the (apparently similar) cultures of those authors and interview subjects which I have incorporated herein.

Terry Winograd describes contextualism and contextualist designers:

> *For the contextualist, usability has its basis in the phenomenology of the user's experience of usability … Because experience is internal and unobservable, the usability engineer cannot know the user's experience of a system through observation alone … Through mutual interpretation the usability engineer and user come to a shared*

> *understanding of the participant's experience of the product … The users become co-researchers … The designer and user enter into a hermeneutic dialogue, in which they are merging distinct horizons and generating a new understanding*
>
> (Winograd, 1995, my emphasis).

Winograd invokes phenomenology to foreground the *intersubjective experience* of usability. *Intersubjectivity* is the phenomenologists' concept of the shared framework of meaning which people employ to grasp each other's worlds. Winograd describes how contextualist designers must take into account the (lived) context of use of a technology. By involving directly the humans that use a technology in its design (often called 'user-centred design' (see p. 118)) Winograd's approach stays true to the subject/object balance required of constructionism. Contextual constructivism can thus be brought to bear on both understanding and design.

Characterising Creative Situations and People in This Research

Recall Richard E. Mayer's clarifying questions for researchers of creativity, presented in the Literature Review. Tom Hewett et al., in the NSF-sponsored report on Creativity Support Tools (Hewett et al., 2005, pp. 10-24) concur that researchers should provide their answers to these questions, and add one more of their own. By answering these questions, researchers locate their research in a specific area of creativity. This is useful in order to align one's findings with those of other researchers in similar fields of creativity, to compare methodologies and metrics, and to appropriately exploit previous findings. I also need to specify the qualities of creative 'situations' and 'people' included in my research question.

To this end, I will characterise the creativity found in the artist-technologist collaborations I hope to support, by answering the questions in the report:

1. *Locus of creativity——is this creativity a property of people, artefacts, and/or processes?* This is an epistemological question, and as a contextual constructionist I take the view that this creativity is a

property of people together with their interactions with the world around them.

2. *Relevance of creativity—is this creativity a personal, social, societal or cultural phenomenon?* Because of the collaborative nature of this creativity, and, to an extent because this is contextual research, creativity is at least personal and social. Depending on the success of the collaboration, creativity is potentially also socially and culturally relevant.

3. *Frequency of creativity—is this creativity a common or rare event?* Creativity is a common event in this research, at least because of the professions of the collaborators (artist and technologist).

4. *Importance of Experience—is this creativity of novices or experts?* [This is Hewett et al.'s addition to Mayer's list] As an interdisciplinary collaboration, creativity is both, because artists may be novice programmers, programmers may be novice artists. The collaboration may enter realms where neither is expert.

5. *Specificity of creativity—is this creativity domain-general or domain-specific?* [This dimension is not in Hewett et al.'s list, but it appears in Mayer's.] I would say domain-specific, on the assumption that domain-specific creativity a) requires some domain-specific knowledge and b) requires domain-general creativity too. Additionally, I assume that artistic creativity is different from scientific creativity, due to the different knowledge structures and tendencies towards different personality traits (p. 32 of this thesis).

6. *Measurement of creativity—is this creativity qualitative or quantitative?* Since the variety of styles of creativity under study is not amenable to psychometric testing, and the timescale is not amenable to reliable 'testing of time', a qualitative treatment of creativity, in real-world contexts, is necessary.

7. *Sociality of creativity—is the creativity individual or collaborative?* Both, for reasons that are hopefully obvious.

In summary, the creativity support tools in my work are designed to support the processes surrounding individual and collaborative, highly-

innovative creative work, in real-world situations, of both novices and experts.

Strategic Approach

Hewett et al. summarise Mayer's (1999) review of research methods that have been used to study creativity, pointing out the strengths and weaknesses of each. In brief, these are psychometric, experimental, biographical, biological, computational and contextual methods. According to the review, since I have identified real-world creativity as the topic of interest, then psychometric, experimental and biological methods would be inappropriate, and contextual and biographical methods would lend themselves naturally. However, quoting from the report:

> *Mayer notes that [biological methodologies'] strengths derive from the fact that carefully documented histories can provide both detail and a feeling of authenticity. However, not all histories are carefully documented at the time events are taking place. Furthermore, potential biases introduced as a result of a focus on a small set of pre-selected people.*
>
> (Hewett et al., 2005)

Biographical studies of creativity, including personal (autoethnographic?) reflections à la Rokeby (1998), are, then, relevant, but I am not convinced that a 'feeling' of authenticity is either sufficient or desirable, especially in the light of Crotty's warning, introduced earlier, against 'rampant subjectivism'. Biographical studies may not produce results sufficiently reliable and impartial to design and evaluate creativity support tools, and so I will discount biographical methods as a way of understanding interactive art practice in the everyday. Hewett et al. go on to describe the strengths and weaknesses of contextual approaches:

> *The strength of [contextual] methodologies lies in the fact that they place the study of creativity in a personal, social, societal, cultural and even an evolutionary context. The projects studied are defined by the practitioner and the research studies creativity using research based in actual practice. The weakness of these methodologies identified by Mayer [(Mayer, 1999)] is the shortage of data and of testable*

> *theories based on such studies. Another weakness of these methodologies is related to the sources of strength and weakness discussed in experimental methodologies. The further one moves away from the controlled laboratory situation, the more difficult it becomes to establish a clear unambiguous set of relationships that support valid conclusions. Research based in actual practice often supports many alternative explanations of what happens and how it happens.*
>
> (Hewett et al., 2005, p. 13)

Clearly, contextual methods go naturally with contextual constructionism.

The difficulty with contextual approaches to this research is that the context changes by nature as the research progresses. This makes it rather too difficult to know what methods to use to evaluate whether a tool or art-methodology is an improvement until I know what that tool or art-methodology is. Additionally, it would be foolish to decide how to construct a tool or art-methodology without knowing what it needs to be. For example, if my first studies found that the most useful contribution I could make to supporting interactive art-making was to use pink highlighters for writing with, then implementing and evaluating that change is likely to be an entirely different process than if I had to rewrite an operating system and evaluate that. So although the entire methodology is presented at once here, in actuality I did not decide on the approach for the later studies before having conducted the earlier ones.

So what contextual approaches do we have available? Thomas and Bevan (1995) outline a methodology called Usability Context Analysis, which focuses on the analysis of the context of use of a product through a cycle of understanding, design, building and evaluation. The authors recommend that understanding and evaluation take place through stakeholder meetings, interviews or questionnaires, with an overview of the positive or negative factors of each of these.

The methodology has a definite focus on business situations, with, for example, "cost-effectiveness" and a variety of "teams" being important

factors, which could be foreign concepts in art-making situations. However, the focus on context and the specific methods may be useful.

Murray et al. (1996) describe a suite of methods called FOCUS, which are geared towards an iterative cycle of development and evaluation of user centred systems whilst looking at the wider picture. The FOCUS methods are related to a development life cycle of requirements design, laboratory testing, field testing and delivery. The requirements design methods are following a project management guide, feasibility analysis method, stakeholders and context analysis and task analysis method. Whilst broadly similar in aim, the approach is not appropriate for my study. In particular, the stakeholders and context analysis method seems to be geared towards eliciting information about specified aspects of a context (for example, individual profiles or working environment), rather than arriving at a holistic understanding of the multitude of interactions and roles within a situation. What's more, the approach aims to provide tools to support external tasks, whereas in my work the aim is to provide tools that are to be used in a creative manner—which means that I assume that the ways in which tools are used could change if the tools change.

The NSF authors provide a set of guidelines to help to develop and evaluate creativity support tools. The guidelines are:

> *Step 1: First it is necessary to observe the activities and problems users are having in real time, either through field/ethnographic research, computer logging, or via actual participatory design.*
>
> *Step 2: Next the researcher must gather user requirements for design of a system that solves real user problems or assists them in activities that they need to perform.*
>
> *Step 3: Design and implement a solution. (This can often be done quickly and inexpensively using low fidelity prototypes.)*

> *Step 4: Iterate via a series of evaluation studies that might start out qualitatively but end up being quantitatively examining new tools versus existing practice.*
>
> *Step 5: Repeat Steps 3 and 4 as often as needed.*
>
> *Step 6: Finally, follow up with the end system out in the field longitudinally, with users using the tools to do their real work in real time over an extended period of time*
>
> (Hewett et al., 2005, p. 16).

Notwithstanding Craft's critique of guidelines, presented below, my goal is aligned with the goal of these guidelines: to understand enough about the use of creativity support tools in order to be able to identify supportive existing tools and uses, and to design supportive new tools and art-methodologies, and to check that they are indeed an improvement. However, these guidelines are a stepwise solution to what, in this case, is a very change-sensitive problem. Changing a tool could change the activities that are useful, and vice versa, and this should be taken into account by using a more agile approach. Also, these guidelines do not call for any particular experience or expertise in the researcher/designer, whereas a researcher having broad knowledge to bring to bear on a problem could be useful in this complex situation.

Terry and Mynatt's work (2002) is an example of the types of process described by the NSF group's guidelines. They used semi-structured interviews followed by interviewee demonstrations (Terry, 2005, p. 20) to investigate how people working in the visual arts used computer tools. The analysis is brief, and the analysis process is not described in Terry's thesis (a results table appears on pp. 30–32), but the most significant finding was that there was little support for comparing multiple ways of solving a problem (Candy and Edmonds' *parallel channels* (p. 43 of this thesis)). As a result, Terry and Mynatt developed tools, called *Side Views* and *Parallel Pies,* to address the lack of support, which Terry and Mynatt evaluated using two controlled experiments (using the NASA TLX workload assessment forms[1] and exit questionnaires) and a think-

1 http://www.nrl.navy.mil/aic/ide/NASATLX.php

aloud study. The evaluation methods resulted in well-defined metrics for certain aspects of creative activity (depth of exploration, backtracking, etc.).

The field study and interview methods that Terry and Mynatt use for identifying the requirements of a creativity support tool are particularly relevant to my work, but the evaluation methods are less so. To evaluate the software, Terry and Mynatt engaged subjects in a specific design task; something that is almost impossible to reproduce in a real-world interactive art project, and measured subjective workload, which does not seem to be relevant.

The use of contextual methods to study art collaboration is also exemplified in the COSTART projects (Candy & Edmonds, 2002; E. Edmonds, Candy, Fell, Pauletto, & Weakley, 2005), which, in the first iteration, paired artists and technologists and analysed their reports, and in the second iteration used trios of artists, technologists and observers, the latter of which introduced a less partial perspective to the reports of the collaborations. The intensively-studied creative work was a series of 5-day residencies (one such residency for each collaboration), situated within a larger staged process of preparation and analysis, with the aim of producing both artistic outcomes and research outcomes. The reports produced from these residencies were combined with interviews and coded according to a formal scheme to illustrate particular, pre-specified differences and commonalities in creative collaborations. The COSTART projects were large-scale, taking place over several years, and with the involvement of many varied groups of highly-skilled people.

In my study, the scale is of necessity smaller, but I can capitalise upon the contextual groundwork laid by COSTART. The important difference here is that I do not wish to impose hypotheses in the form of pre-existing coding categories of tool use, as the scale of my study is small and qualitative enough in nature for it to be easy for me to seriously skew the results with such prejudice. Moreover, the production of artwork is not a necessary direct outcome of my study.

A NOTE ABOUT PRELIMINARY STUDIES

The foundation studies, developments of two artworks, were conducted as part of other projects, so the methodologies used in them were not

particularly appropriate to this study. However, it is still possible and useful for me to informally describe and reflect upon them in order a) to convey something of what it is like to develop an interactive artwork and b) to touch upon some issues which become important in later studies.

Exploring Tools

Since, at this preliminary stage, we have only identified an area of interest, with no particular goal in mind, we can use exploratory methods to refine that area and decide upon more formal methodological requirements. It seems sensible, before embarking upon any particular observation of social processes, to have a broad understanding, through practice and reading, of the state of the art in the design of creativity support tools, in order to see whether ideas from one tool area could be applied to problems in another tool area. This approach would have the additional benefit of allowing the researcher to understand the technological scope and impact of a change. For example, a researcher may not realise that a need for tools which was highlighted through observation alone may be symptomatic of a wider technological need; and so may end up misapplying effort to solve a too-small problem (a "hack") (for example, the programming system *Processing* was described by its creator at a panel session (Reas & Puckette, 2004) which I attended as, paraphrasing, "not a vision of the future, but a good, thorough, hack"). Conversely, improving one area of creativity support may have a negative impact in others, which would have been foreseen by a knowledgeable designer.

Of course, argument from practice and reading is too subjective to reliably produce effective design or to form the entire basis of constructionist research. On the other hand, some historians have made personal observations and drawn conclusions about early-stage technology (perhaps because technology's scientific realist background cries out for subjective interpretation) that have been very well-received and influential in future design. A canonical example is Shneiderman's "Direct Manipulation" paper (1980), in which he relates his (relativist) "admiration" of users' "delight" for certain "lively, enjoyable interactive systems", despite in the same section having described (as a realist) that researchers are improving "human engineering" by "applying controlled, psychologically-oriented experimentation to develop a finer under-

standing of human performance" (p. 68). The novelist Neal Stephenson has written an essay called "In the Beginning was the Command Line" (Stephenson, 1999), Which charts the rise and future of operating systems, which has been well-received by open-source communities.

There is, then, still a place for reasoned arguments, based upon an experienced individual's (constructionist) interpretation of history, to analyse and influence the path of interface technology development. I do not propose to say, however, that whatever anyone argues should be taken as gospel. In fact, having made this argument from literature, I myself produced recommendations for technology to support interactive art making, but had no way to tell the relative value of any of these recommendations and so could not produce from them any specific design which took into account interactive art making's lived and social contexts of use. I realised that although my newly-acquired expertise could prove useful to bring to bear on the research problem, what was still needed was a rich theory to help to understand of the social and art-methodological impact of the technological themes of my argument.

Exploring Art-Methodology

Various methods exist to t*est* social theories, but what was needed at this stage was a way to *generate* a social theory that is responsive to the social situation of making interactive art. Two families of methods that conventionally apply to this problem are action research (e.g. O'Brien, 1998) and grounded theory (B. G. Glaser, 1992; B. G. Glaser & Strauss, 1967; Strauss & Corbin, 1998), which share similarities. Action research is distinguished by its dual aims of action (change) and research (understanding), the first of which is not required at this stage. Action research also tends to be participatory (that the researcher is involved in the situation under study), which means that, used on its own, rigour may be an issue to a wider audience.

So whilst action research will become more relevant later on when I make changes to the situation, at this stage we only seek to understand the situation from an external point of view, and to achieve this goal, the use of grounded theory is appropriate.

I found independent parallels with my decision process in other design research. Reinvoking Shneiderman, Brock Craft (Craft & Cairns, 2005) has studied citations of Shneiderman's paper "The Visual Information-Seeking Mantra", which is one of several papers in which Shneiderman make recommendations for effective interaction design based on argument. Craft notes, however, that designers tend to assume that, since they followed the guidelines in the mantra, the visualisation was effective, even though "there are no reasonably obvious studies that have validated Shneiderman's recommendations" (p. 110). Craft highlights the need for a robust [design-]methodological framework for such guidelines, to unify their recommendations so that they are consistent with one another, and to clarify the rationale, which includes examples where the guidelines are known to work (and are known not to work). Craft calls for a "holistic design methodology", with which design patterns can overcome their own limitations, including those of subjectivity. Craft's proposed approach to finding such a design-methodology is also based upon grounded theory.

As Kautz et al. relate,

> *The literature on [Information Systems development methodologies] is extensive and wide-ranging. It consists however largely of prescriptive and normative textbooks and work that is based on anecdotes, but there is limited scientifically collected and analysed empirical documentation…*
>
> (Kautz, 2004)

In other words, there is a growing need to understand software development from a social perspective. Kautz et al. go on to provide an illuminating grounded theory study (Kautz, 2004; Kautz, Hansen, & Jacobsen, 2004) of actual use of development methodologies in one large company, to counter this state of affairs.

Grounded theory is a methodology that can produce surprisingly rich theories to explain social phenomena, from little more than repeated careful observation (imagine conducting a scientific experiment where no hypothesis pre-exists, and no variables are controlled), and it has

been employed in a variety of social contexts, including this one. The particular advantage of grounded theory is that every aspect of a resulting theory can be split into multiple sub-aspects, and finally to actual observations. This provides for great richness and concreteness of theory, usefully systematic comprehension of data, and as such makes it harder for the researcher to insert his or her preconceptions. I have conducted a grounded theory study because I felt it important to study the social interactions across and between two disciplines during art-software development, so that methodologies and technology can be developed to support those interactions. As a result, I was able to discover a rich theory about the correspondingly rich methodologies used in interactive art development. By testing the theory, through production and evaluation of tools that support it, we can then form methodological recommendations for practitioners. There is no reason to suspect that social study, and grounded theory in particular, would be any less useful in developing useful methodologies and supportive tools in other software engineering contexts.

INTRODUCTION TO GROUNDED THEORY

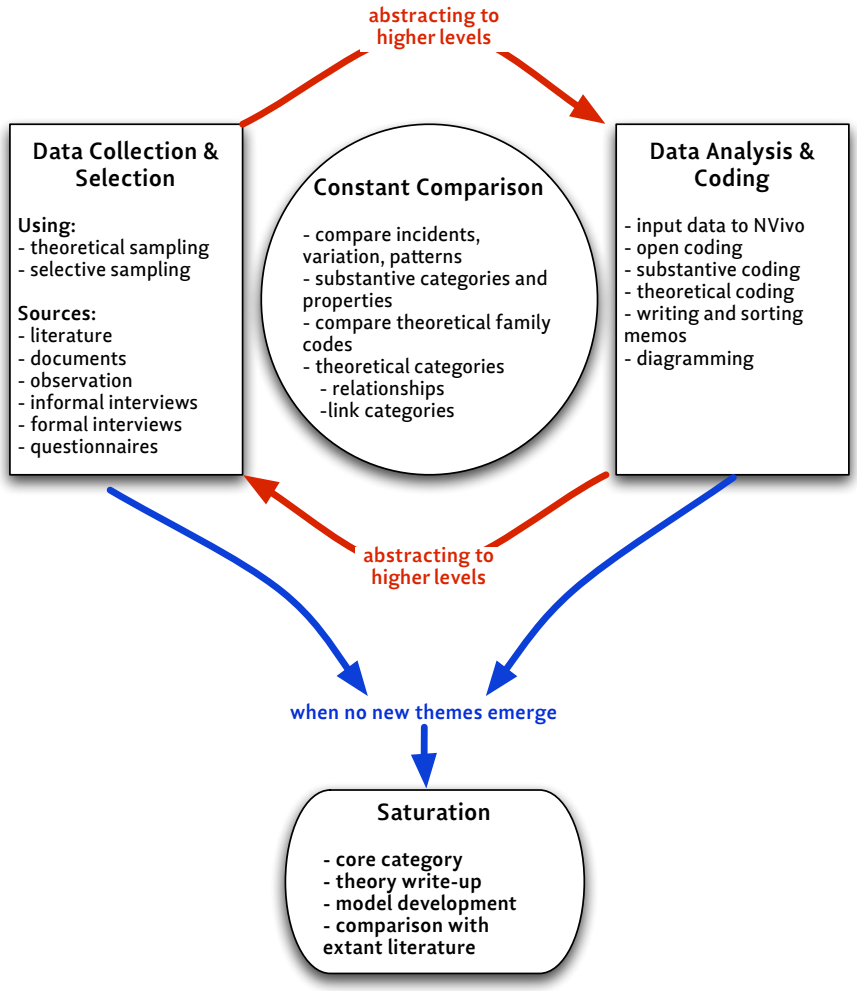
Grounded Theory is an approach where the theory emerges from the data itself, and is thus grounded in it, rather than being an approach which tests existing theories in the form of codes or otherwise. The theory's emergence from the data means that any bias or preconceptions held by the researcher should have minimal impact. Grounded theory arises as a result of constant comparison of incident with incident (and incident with concept), which builds up concepts and thence theories.

There are two main schools of grounded theory analysis, those of Anselm Strauss (Strauss & Corbin, 1998) and Barney G. Glaser (B. G. Glaser, 1992), erstwhile colleagues, and the two originators of the approach, in a study on the social understanding death in hospitals (B. G. Glaser & Strauss, 1967). The two have since disagreed (or rather did until Strauss' death in 1996) on whose version is most correct. Having appraised both, assisted by Babchuck's examination of the controversy (Babchuk, 1996), I have chosen to follow Glaser's approach, since it appears to be more elegantly simple and with a greater focus on emergence of the theory implicit in the data. Simplicity, combined with Glaser's convincing (yet, alas, acrimonious) critique of Strauss and Corbin's question-

asking process, means that I think Strauss and Corbin's version is more prone to distortion by the researcher and is hence better suited towards theory verification than theory generation.

GROUNDED THEORY PROCESS

Grounded theory begins with no preconceived codes, and produces codes from the iterative asking of neutral questions. Kan (2002) represented the process as a diagram, redrawn here.



A small yet exhaustive list of questions to ask of the data, for all coding stages (taken from Glaser (1992, p. 51)), follows:

- What is this data a study of?
- What category, or what property of what category does this incident indicate?

- What is actually happening in the data?
- What is the basic social psychological process or social structural process that processes the main problem that makes life viable in the action scene?

The first iteration of grounded theory analysis is the open coding stage. Open coding (in which 'every' bit of data relevant to the field of study is coded) focuses on the establishment of broad categories, and hence on the asking of the second and third of these questions. The open coding stage can draw from widely-varied and general sources, so long as they belong in the social situation under study. Examples are generalised questions or observations, or existing case studies. Literature reviews or other socially 'external' sources should be introduced later; used initially, they are prone to distort the subsequent analysis. Glaser then advocates several iterations of gathering more data, constant comparison and selective coding (coding the data which resonates most strongly with existing codes) in order to establish categories and super-categories of codes, and to situate the most important categories within a framework that shows the overall picture.

Constant comparison is at the heart of the grounded theory process. Items of data are compared with other items of data and similarities or differences between them can contribute to codes. Themes found in the codes contribute to categories.

Categories that get 'saturated' with data points (saturation occurs when more data would not contribute new meaning to the category) are more important to a theory than categories that do not.

The categories are linked with 'memos', the researcher's speculation over possible links between categories. Memos associated with saturated categories can go on to form part of the grounded theory, or can be superseded by further memos as new aspects of the theory emerge.

INFLUENCE OF PRECONCEIVED IDEAS

Since, in Glaser's grounded theory, the theory emerges from the data rather than being imposed upon it, Glaser encourages grounded theory researchers to remain open, and to avoid commencing the research with

preconceptions. This avoidance extends even to the literature review process—grounded theory researchers should avoid embarking upon a literature review at the outset. This is the case even though, as in this research, the literature review is conventionally undertaken at an early stage, and before deciding upon methodology in order to develop the theoretical framework in which to consider the problem.

However, the elimination of preconceptions is impossible in practice, as any researcher will inevitably have ideas or beliefs, no matter how rudimentary or subtle, that will influence the emergence of theory and the research direction. Moreover, since I have carried out both a literature review and an argument from literature before deciding to conduct a grounded theory study, I am at a greater risk of being influenced by preconceptions. To mitigate this risk, Glaser (B. G. Glaser, 1998, p. 120) suggests that the researcher could:

> *write a paper on his literature review and publish it, in order to establish and state the assumptions he absorbed from the literature so they become data to constantly compare with what is really going on.*
>
> (B. G. Glaser, 1998, p. 120)

I have published such a paper (Turner & Edmonds, 2003), included in Appendix 1, and introduce its contents as data for comparison during the study.

COSTART Data Analysis

The preliminary source for open coding was the collected Case Reports for the first COSTART (COmputer SupporT for ARTists) project, mentioned earlier (Candy & Kelly, 2000a, 2000b, 2000c, 2000d, 2000e, 2000f, 2000g), which is appropriate because it centred round seven intensive artist–technologist collaborations. I coded Artist, technologist and observer statements, from recordings, interviews and diaries, by hand, and excerpts with the coding appear in Appendix 3 (on CD), Section 1. (The complete original coding is not included for confidentiality reasons, and because large tracts of the source material did not contain codes that were corroborated in other reports—in other words, they were unique to that situation). These artists and technologists worked together in collaborations leading to and including an intensive one-

week residency, during which the interviews were conducted. The level of coding at this stage is relatively cursory, because the reports are non-primary data and it was the intention to get a very broad feeling for the issues involved.

I arranged codes from the first iteration of coding according to artist and technologist, and so produced a list of categories, which appears in Appendix 3 (on CD), Section 2.

More open codes were taken from primary data—transcripts of interviews with artists and technologists from the larger and more meticulous COSTART 2 project (Candy & Edmonds, 2002), which involved 9 further residencies, bringing the total to 16 artists, 6 technologists and 4 observers. These were coded using NVivo software (Fraser, 1999), and this coding appears in Appendix 4 (on CD), Section 1. NVivo is designed for coding text, and is a computational way of managing the documents and lists of codes and memos that would otherwise take place by hand. At this point there was a list of categories indicating the many and various issues that arose and were recorded during the collaborations, which Appears in Appendix 4 (on CD), Section 2.

INTERVIEW ANALYSIS

I needed to decide upon an appropriate way to gather some of the data needed to saturate some of these categories. I could have conducted an ethnographic study, observing art practice à la COSTART; but even COSTART ethnographers could not observe all aspects of a creative collaboration; carrying out such a large-scale study would be overkill simply to saturate codes. Instead I chose to conduct a series of qualitative interviews with artists and programmers who had been involved in collaborations.

Interviews are a recommended data gathering technique in grounded theory, because it is a commonly used for technique in social research, and it is applicable here (Foddy, 1992). According to Foddy, interview researchers have traditionally made the following validating assumptions:

1. The researcher has clearly defined the topic about which information is required.

2. Respondents have the information that the researcher requires.

3. Respondents are able to access the required information under the conditions of the research situation.

4. Respondents can understand each question as the researcher intends it to be understood.

5. Respondents are willing (or, at least, can be motivated) to give the required information to the researcher.

6. The answers that respondents give to a particular question are more valid if they have not been told why the researcher is asking the question.

7. The answers that respondents give to a particular question are more valid if the researcher has not suggested them to the respondents

8. The research situation per se does not influence the nature of the answers given by respondents

9. The process of answering questions per se does not change the respondents' beliefs, opinions, habits, etc

10. The answers that different respondents give to a particular question can be meaningfully compared with one another.

However, each of these assumptions is more complex than they may at first appear, and that care should be taken to avoid hidden pitfalls in oversimplifying these assumptions. In order to distinguish the feelings and actions of people in the artist role from the feelings and actions of people in the programmer role, and to explore the issues facing non-programmers, initial subjects were selected who were interactive artists who do not program, and programmers who program for such artists. The interviews were conducted in December and January 2005, face-to-face (with one exception where iChat was used), were recorded and, for the first iteration, transcribed for coding in NVivo. Six subjects were interviewed—four programmers and two artists who had worked with these programmers, in order to present 'both sides of the story' for analysis. The subjects were respondents to a personalised emailed call to

around fifteen artists and programmers in Sydney who were known to my colleagues or me, word of mouth was also encouraged.

The interviews were semi-structured and qualitative, with questions designed according to best practice as described above, based upon the categories that were popular from the COSTART analyses, but as yet unsaturated. The designed questions appear in Appendix 5, and are justified in Chapter 6, but were adapted and extended to suit the individual conversation—the adaptations and extensions appear in the transcriptions.

The lengths of the interviews ranged from 30 minutes to 1 hour, apart from the iChat interview. The iChat interview was carried out as a pilot technique, both for testing the questions' interpretation in a non-rich communication medium and to test the applicability of the communication medium itself. It produced worthwhile results for coding, and did not need to be transcribed, but after that pilot I felt that iChat did not convey emotional information as richly as face-to-face interviews, and took longer (1 hour 45 minutes).

With the exception of the iChat interview (the transcription of which was the log of the iChat software), the interviews were recorded directly into an Apple Aluminium G4 PowerBook using the computer's built-in microphone. The software used was *Audio Recorder* (Shanfelder, 2005), which saves the files as MP3s. The MP3s were transcribed using Express Scribe (NCH Swift Sound, 2004) by paid transcribers. The transcription documents were then imported into NVivo for coding. The questions appear in Appendix 5 and the transcripts of the interviews appear in Appendix 6 (on CD).

The codes resulting from the interviews were arranged into the categories that had previously emerged in order to see if they saturated those categories. Where some categories did not seem to be saturated, even though questions covering them had been asked, I re-examined the contents of those categories and in some cases incorporated them into other categories, and in other cases left the category as-is, unsaturated, and discounted it from the theory. A full description of this process is in Appendix 7, Section 1.

Arrangement of the categories into super-categories was done mostly by hand, and is reproduced in Appendix 7, Section 2.

After the several iterations of selective coding and data gathering, a hierarchy of about 200 codes and categories of codes emerged, with associated memos describing the relationships between codes according to grounded theory analysis. At this point, enough categories were saturated to form the super-categories: Relating to the Project, Collaboration Patterns, Developing Problems into Shared Structures, Artists Exploring Technological Structures: Naïve Interactive Art, and Human Computational Interfaces, Technologists Exploring Artistic Structures: Intimate Iteration and Computational Toy-Making. The memos from the study were collected to form the theory described in Study 2 (Chapters 6 and 7).

Evaluation through Collaboration

To reach the final stage, I have made an argument that identifies long-term technological requirements for supporting the creation of interactive art, and I have used grounded theory analysis to identify the more-immediate technological needs and related methodology. Together these studies have given us a framework for deeper understanding of interactive art creation, and it remains only to apply that understanding in order to evaluate the framework. We can demonstrate such applications by developing interactive artwork that uses the framework, and evaluating that development process. So we have two outcomes—firstly the artwork itself, which, whilst useful to evaluate in terms of success in its own right, does not tell us a great deal about the quality of the development process. It is more important to deeply evaluate that process, through further interviewing and observation, and matching the subsequent analysis against the recommendations of the framework.

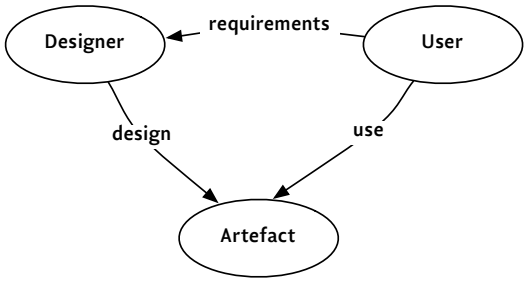
The findings of the foundation work (Chapter 4) and the first study (Chapter 5) call for a major reworking of software, which would be the product of a long-term research endeavour. The second study (Chapter 6) identified the need for tools and methodologies to make and use computational toys in order to share meaning; a goal much more easily realised in the short term.

The long-term scale of the implications of the first study mean that it is only possible to produce high-level recommendations, and to evaluate those recommendations by implementing low-fidelity prototypes, or to use the promising leads in existing technology, as identified in that study.

Systems which help to make computational toys, as suggested by the second study, are easier to envisage, yet, in order to evaluate them, they must exist within a real-world art endeavour. Therefore, I sought an artist who was willing to use one of the leading technologies that I had identified in the first study, and have me create prototypes of technology and methodology that aligned with the findings for him or her to use in art development. The success of the artistic outcome, and, more importantly, the views of this artist, would be a means to evaluate the usefulness of the computational toy-making technology, and for generating deas for future research.

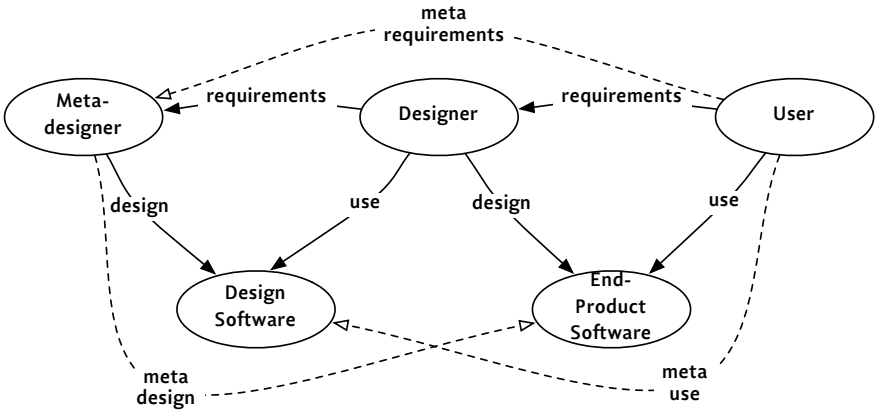
SOFTWARE DEVELOPMENT

As suggested by the second study, my intimate iteration software development methodology was used in the collaboration, similar in outlook to Extreme Programming (Beck, 1999) and other agile software development methodologies. The goal was also to make (or augment) a creative environment for the artist to produce the artwork (with the aid of 'computational toys', rather than for the technologist to directly assist in the artwork's production. This creation of an environment is an example of Fischer's 'metadesign', and I have adapted the agile method of User Centred Design (UCD) (Norman & Draper, 1986) to work in metadesign. This is a diagram of the conventional UCD approach:



In UCD, the user feeds requirements to the designer, who designs the artefact in response. The user uses the artefact and more requirements are given to the designer, who improves the design, and so on.

In metadesign collaborations it is important to distinguish between two levels of 'user', and hence two concurrent levels of user-centred design. The first level, most relevant to my research, is the a*rtist* as the user of the technology and methodology for creating interactive art. The second level of user is the a*udience* as users of the art. The user-centred design system for metadesign looks like this:



There are two conventional UCD triangles in this arrangement: on the left, the artist's design, with audience requirements and use, and on the right a metadesign, with the artist's requirements and use. From the metadesigner's perspective, the audience requirements are metarequirements—that is, the requirements of the requirements-maker. The complicating factor in gathering requirements for the metadesign is that the metarequirements originate from the audience, but, due to the artist's own involvement in the gathering of those metarequirements, he or she is able to mediate them and incorporate them into his or her own requirements.

Because the art collaborations I studied incorporate this process, it is also incorporated into the intimate iteration development art-methodology, which emerged from the study, and which I used in the new collaboration. To introduce it briefly here, intimate iteration was identified as being a common, though far from universal, technique for interactive art development. In contrast with traditional iteration, where

the programmer works individually and separately according to iterating specifications, intimate iteration takes place with the artist and programmer in the same room, often, side-by-side, with the programmer modifying the program in direct consultation with the artist. Intimate iteration is advantageous because it allows the artist to more closely control the development of the software, but the process can be frustrating (in terms of speed and conflicting approaches) for both parties.

INTERACTIVE ART EVALUATION

The evaluation of the art itself is secondary to our aim of supporting the art-making process. However, it is less convincing to claim that the collaboration process was successful if the result of the collaboration is not successful, so we will deal with the topic briefly here. The evaluation of the success of interactive art, (as with art in general) is far from straightforward. The standard approaches for evaluating interactive systems come from Human–Computer Interaction (HCI), and are generally geared towards improving user satisfaction or task performance. In art, however, the object is not always to 'support' the user in this way—on the contrary, examples of interactive art exist where the user is meant to become frustrated (Höök, Sengers, & Andersson).

Lizzie Muller has developed a pioneering approach, combining techniques from the fields of art criticism, curatorial practice and HCI. The approach aims to realise the affective goals of the artist as closely as possible by studying the *audience experience* of the visualisations and incorporating the findings into an iterative design process. (E. Edmonds & Muller, 2006; Muller & Edmonds, 2006) Muller carried out an evaluation of the work we created using this approach, and her results are presented in the evaluation chapter (Chapter 9) and published in Khut and Muller (2005) and Muller, Turner and Khut (2006). Having a third party evaluate of the artwork, through the work's audience, lends additional credibility to an assertion of the collaboration's success.

COLLABORATION PROCESS EVALUATION

Recall the comment of Resnick and his colleagues in the NSF C*reativity Support Tools* workshop report:

> *It is still an open question how to measure the extent to which a tool fosters creative thinking. While the rigour of controlled studies makes them the traditional method for scientific research, longitudinal studies with active users for weeks or months seem a valid method to gain deep insights about what is helpful (and why) to creative individuals*
>
> (M. Resnick et al., 2005, p. 34)

As I have said, the evaluation of this work needs to take place in a real-world art context. The artist's (admittedly subjective) evaluation of the toy-making methodology should be sought; the ability to respond to desired/requested changes should be observed. My (again, admittedly subjective) evaluation of the work should be included. Finally audience feedback should be taken into account to judge the overall success of the work.

Porteous et al. (1993) present the Software Usability Management Inventory, which is a questionnaire that measures "how usable a software product is according to the perceptions and attitudes of users" (p.1). It is useful because it focuses on the feelings that a user has, rather than some objective measure such as task completion. The method is useful as a guide of what aspects of software produce feelings (e.g. taking too long to respond, remembering how to do things). However, the method requires a sample of 10 or more users per system to get meaningful results, and those aggregate results would not well represent the richness of understanding of unique cases still needed at this stage of research to validate the grounded theory. Furthermore, it treats a single piece of software as a hermetic system, rather than a component of, or addition to, a wider software development environment, and particularly when that environment is itself a software development environment (refer to the diagram showing user-centred design system for metadesign). For example, one statement in the survey is "learning to operate this software initially is full of problems". In the software in my study, this question could refer to the process of learning a new tool's host software, Max/MSP, or problems with the connected software or hardware, which could, in a development environment, interfere with the tool learning.

At this stage of technology and methodology research, we need rich evaluation of processes more than we need statistically-unassailable yes/no answers. Since I am also evaluating a rich theory, it makes sense to use the same techniques to interview the artist, and compare the new results with the theory's results.

I initially considered having someone else, not myself, conduct the interview, because I was aware of the potential pressure on the artist to be supportive of my research, and of the risk of embarrassment about talking about our collaboration in a critical manner. However, unless the artist was to be deceived by the interviewer, it would be clear that the interview would sooner-or-later be analysed by myself, so the net effect on the artist's answers would be small—this belief is corroborated by a seeming lack of such a procedure in the literature. Besides this, I have the advantage of being more familiar with the collaboration than a third party interviewer, so would be able to interpret answers more richly during the course of the interview.

Summary

The methodology I have chosen here has been shown to be valid and useful to tackle this problem, as supported by the work cited, but as far as I know, this combination of methods is the first such combination. Whether this methodology is more generally useful, useful for similar problems or can be extended further is largely a question for future research. The power of grounded theory in particular, however, is that all the separate concepts and codes in related studies can eventually be built up into 'super-theories', providing a rich and complex understanding of human practice in a broad area.

In this project, different methods were used to understand (historical and technological) context, to understand social behaviour within the context, to design and make the solution and to evaluate the solution. Together, the methodologies and their results allow for a rich understanding of the multitudinous and multifaceted issues in this creative situation.

The research question is this: *In a situation which otherwise supports creativity, what would be useful tools and related methodologies to help people create*

interactive art? To begin to answer this, I need to make sure that I have grounded expertise in the tools and techniques that already support people to create interactive art. Although the literature review identifies desirable qualities of these tools, in order to become such an expert, I must have surveyed the state of the art in these tools and techniques, and must have made some art myself using some of them. I must reason about these processes in order to identify, along the lines of Shneiderman, what I think are the areas in tool design most lacking.

However, being a conscientious constructionist, I cannot assume that my expertise is sufficient to allow me to design such tools just by thinking about them. Grounded theory allows me to take this literature-derived expertise and to build upon it using an iterative process of questioning, data-gathering and coding analysis. What's more, this process does not presuppose any particular scheme for organising results, and allows the most significant results to emerge as such in the analysis.

Having identified the most significant design aspects for new creativity support tools, it is time to create these tools. Again, this process cannot happen in isolation, and so I developed these tools in the context of the development of a real-world art development. This development was evaluated from several angles—the artist's, the curator's, the audience's, and my own.

4. Preliminary Studies

Introduction

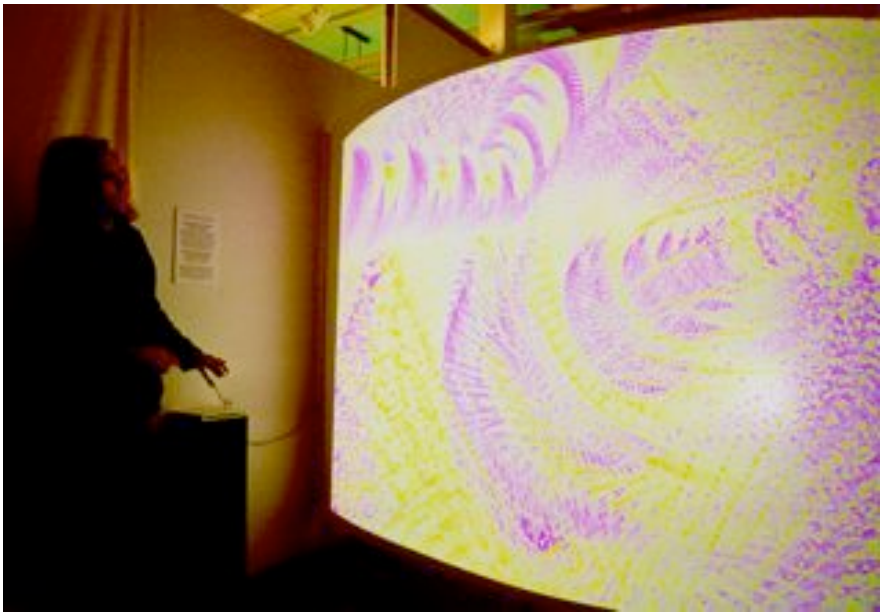
This chapter describes the art collaborations I have been involved with which have influenced my work in the Ph.D. Another piece of art has been a formal part of the work on my thesis, and this is described in Study 3 (Chapter 8). The pieces described here have been influential in my research, so it is relevant to include them here. This is firstly so that the reader who is inexperienced in interdisciplinary artistic collaborations can understand the process in more detail from reading about specific cases and secondly to contrast artworks which took place before my studies with the evaluative artwork at the end of them.

I have described two collaborations in this chapter, in chronological order. Biographies of the artists appear in Appendix 10 (on CD). The first collaboration, *cubeLife* with Dave Everitt, is my first artwork, and inspired me to do a Ph.D, in this area. This piece is written in Java, and the case is an introductory example of a situation where tools for making computational toys would have been extremely useful. The second collaboration, *Séa.nce* with Norie Neumark and Maria Miranda, is interestingly self-referential because it was a collaboration across geographical

distance, to make an artwork which itself deals playfully with collaboration over geographical distance, which tells us things about the roles of technology and face-to-face communication in collaborations. Moreover, the *Séa.nce* development is an example of encountering the frustrating constraints of an application that is designed for making interactive art. A third artwork, *Cardiomorphologies v. 2*, is explicitly a part of my study, and is described in Study 3 (Chapter 9).

cubeLife

cubeLife is a collaboration between artist Dave Everitt and myself. The current version of the cubeLife artwork is based in a networked virtual world populated by magic cubes. Each magic cube is created by participants' input; either on-line or in the exhibition space via a fingertip heartbeat sensor. Each cube has a finite life and associated sound, and inhabits an artificial life environment where it can be made to flock with other cubes and follow various defined behaviours. In this piece, the warmth of the human heartbeat is juxtaposed with the strict structures of mathematics and artificial life.

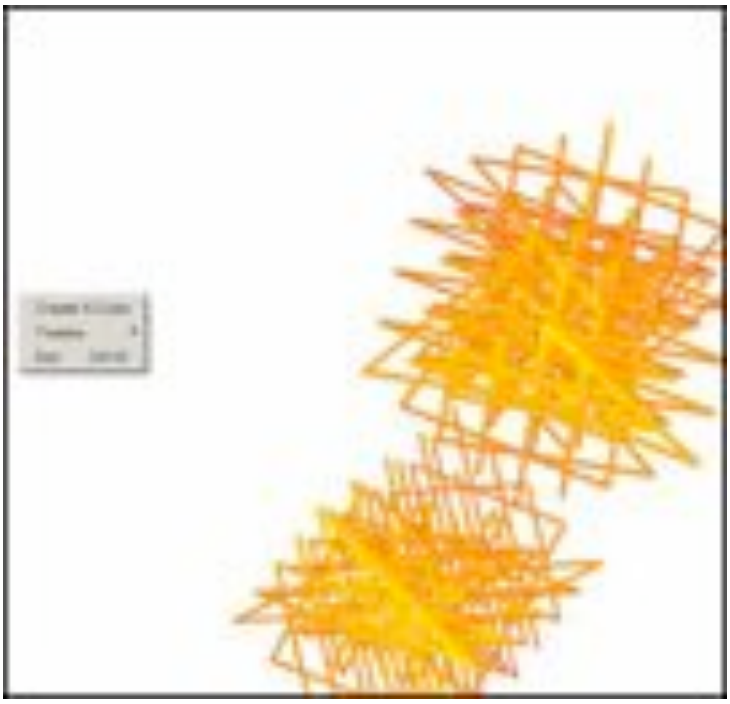


cubeLife was constructed in 1999–2002, was exhibited at the 1999 bioMatrix show[1] , and is permanently online (http://www.cubelife.org), with a minor update in 2005 for an exhibition in Derby UK[2] . *cubeLife*'s development was the topic of my Master's thesis in Computer Science (Turner, 2002). That thesis was a technical report of research and development from a fairly traditionalist computer scientist's perspective,

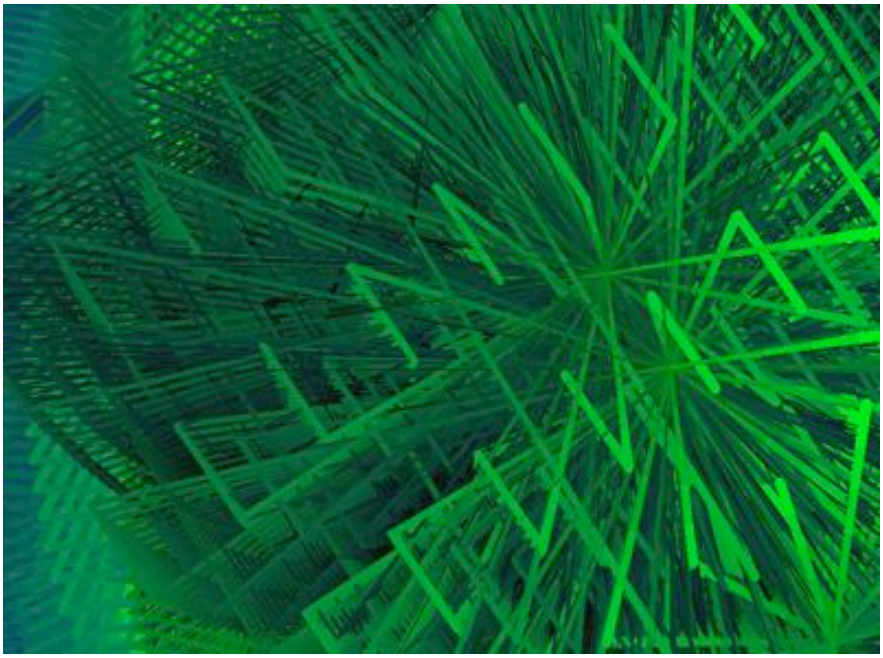
1 http://www.daveeveritt.org/matrix/matrix.html

2 http://www.q-arts.co.uk/parseTemplate.asp?id=169

which I will not repeat in full here. Instead I shall reflect upon the experience of my first artistic collaboration from the point of view of the current work.

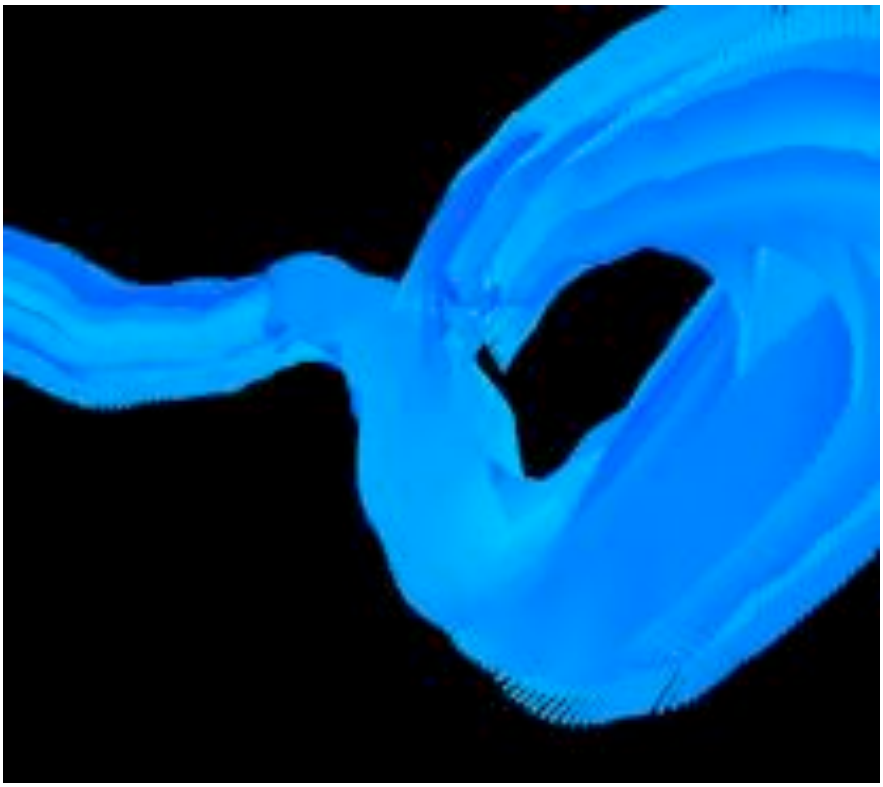


Primarily, the thesis report is characterised by a tension between traditional computer science determinism and the unknown excitement of working on an artistic project. For example, the development begins with agile-methods-style metaphors for behaviour, then moves onto a large, traditional, requirements capture phase, then returns to a more agile methodology for the 'intimate iteration'-style development (see Chapter 7 for my description of intimate iteration and Chapter 8 for my evaluation of it). Also, the report contrasts dry technical diagrams with sketches and screenshots. I now recognise this tension as being emblematic of my own change from being primarily interested in the technology of computer science to being primarily interested in its human aspects.



The guiding metaphor we picked for *cubeLife* was this: The *cubeLife* system can be thought of as being like the surface of a pool of water. Anyone can drop stones in the water, and each stone creates a unique ripple on the surface. Each ripple behaves subject to laws governing its motion, and can interact with other ripples. Every observer sees a different view (or the same view differently). I was able to translate this description into three concurrent threads—the process of modelling the ripples, the process of seeing what's happening, and the process of throwing stones. This is a classic model–view–controller system, and I implemented it as such in Java. However, the devil is in the details, and Dave and I attempted to decide as many of those details as possible up-front in the requirements gathering document (there are eleven A4 pages of requirements). This document, a holdover from the days of waterfall-based methodologies of software development (where each phase of development must be completed before the next can begin) was of mixed use. In the first place, *cubeLife* is a reasonably complicated computing system, for a developing computer scientist, and more so for Dave, a non-computer scientist who occasionally experiences the effects of a cognitive disability; we both felt the need to have this structured document to think through all the issues and cover all the bases. The requirements document had scores that Dave could give for both *Satisfaction* (how pleased he would be if the requirement was met) and *Dissatisfaction* (how displeased he would be if the requirement was not met), which helped us set priorities. Discussing the requirements was a useful way for Dave and I to build a shared language. The requirements document was useful while *cubeLife*'s engine was being built, but I am not convinced it was more useful than any other method for developing shared language and setting priorities. Moreover, as we were to find out, it is often impossible to know what you want until you can see it, and in

reality most of the development time was in experimenting with motion models and establishing the precise rules for behaviour.



For example, we developed a small expert system to choose colour schemes, completely unanticipated in the requirements document. I started with a straightforward algorithmic way to generate colour schemes, based on the output of programs like *ColourSchemer*[3] , which basically take colours from equal spaces around the colour wheel, plus variations on hues and saturations. However, the results were not always pleasing, with lots of dark or muddy, or garish and bright colours being created. In addition, there was no scope for variation amongst cubes.

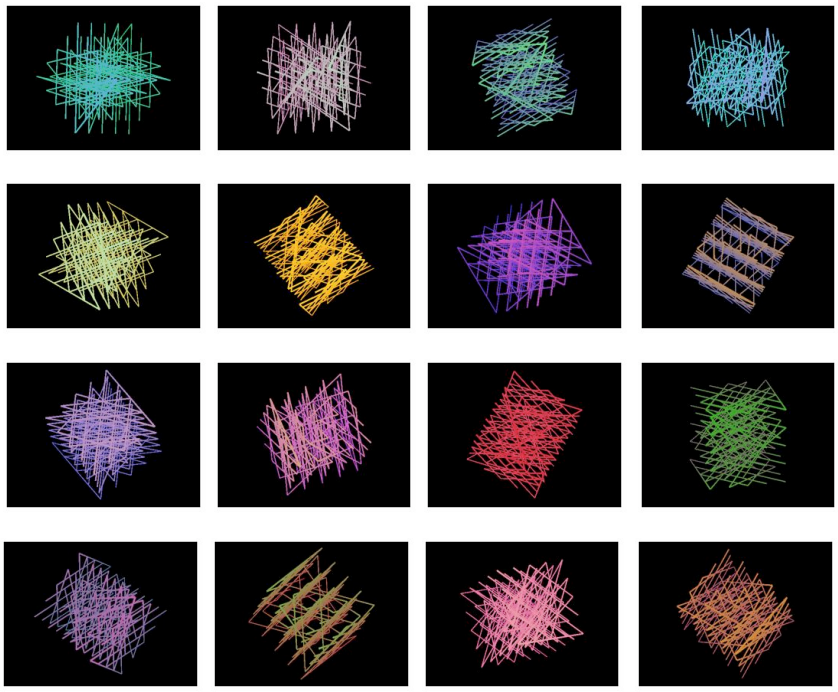
The difference between the scientific method of selecting colours and the artistic one was intriguing, so in a meeting with Dave, I constructed, with his guidance, and with the help of the Apple Colour Picker, some rules which attempted to convey his intuitive feeling for colour sets.

Broadly, the rules operate as follows: A colour with a random hue, and random saturation and value above certain minima is chosen. Then it is tested to see whether it is a nasty shade of green, magenta or cyan, which are the three colours Dave singled out. In the case of the former colour, the range of coordinating colours is limited; in the case of the latter two, the colour is changed to become more pleasing. The background colour is chosen to be within a minimum and maximum hue

3 http://www.colorschemer.com/

distant from the foreground colour, and hue and value are again randomly picked to be above a certain minimum.

When it comes to choosing the colours for a particular cube, small variations on the colour scheme colours are created, to avoid each cube being identically-coloured.



We developed this colour-choosing process in an example of what I now call intimate iteration, in that the system is hard-coded, with parameters in a configuration file. However, it is a perfect example of where computational toy-making would have been useful. Had it been easy to create toy interfaces in hand-coded Java, it would have saved time and added control to build one for this purpose.

Séa.nce

Séa.nce (full title: *Séa.nce: a networked glossalalia*) was created as a collaboration between artists Norie Neumark and Maria Miranda, based in Paris, and myself, based in Sydney, assisted by Alastair Weakley, also in Sydney. After two introductory meetings in November and December 2003 we worked on the whole project online (no telephones or face-to-face meetings), until the work was ready for installation in August 2004. The work was launched on an international ferry travelling in the Baltic Sea as part of ISEA2004 (International Symposium of Electronic Art, which is a prestigious new media art conference).

The purpose of this chapter is to describe the development and semi-formal audience evaluation of a piece of interactive art, so that the

reader can get a feel from examples, both for the process of collaborating over art, and for the technological and interaction issues that art presents.

The content of this section is based upon our paper (Turner, Neumark, Miranda, & Weakley, 2004), which appears in Appendix 1. It is a co-authored paper, due to the combination of views being necessary to present a complete description of the process. I was principal author, and the contributions of the other authors are attributed in the excerpt with their initials (AW, NN & MM) in this chapter. The conclusion has been slightly reworded to be clearer. Broadly, the structure of this section is as follows: First, Neumark and Miranda's background to *Séa.nce* is described, so that you may become acquainted with the concepts, methodology and terminology we employ. Next, I describe the significant artistic, interactive and technological features that emerged during the development of the work. Thirdly, we describe our reflections on the collaborative process and the role that collaboration played in the development of the piece (and indeed vice-versa). Finally, I give my reflections on this project as it pertains to my thesis—the *Séa.nce* development is a typical example of coming up against the constraints of existing applications.

—Beginning of paper extract—

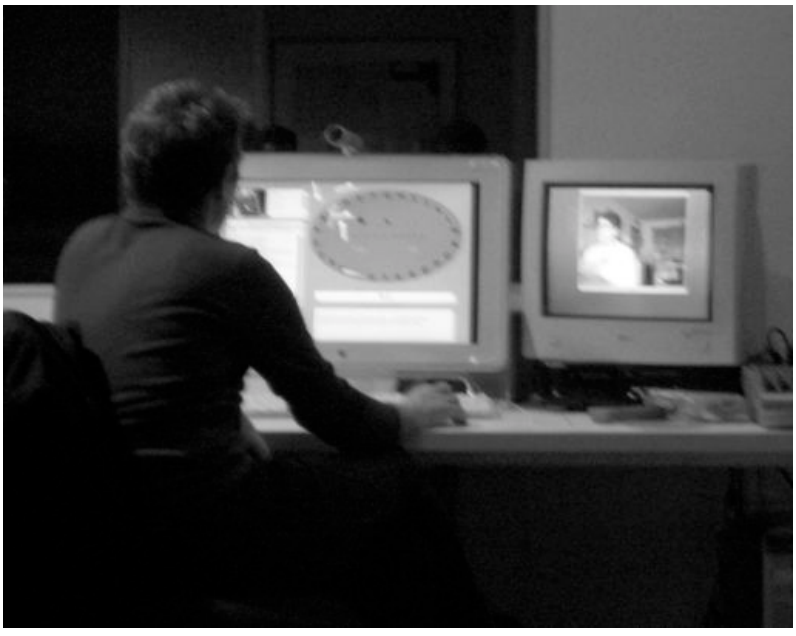


Figure 1. A participant in the semiformal trial of Séa.nce. The right-hand monitor shows the video feed from the artists in Paris.

In writing this paper, we considered three different aspects of *Séa.nce*: the artistic aspect (e.g. the question to be explored), the interaction design aspect (e.g. the desired effect for the audience) and the technologi-

cal aspect (e.g. implementation issues). These aspects are so highly interrelated that their descriptions in such interdisciplinary research are prone to confusion, both for authors and readers, and so we have mapped out a tentative structure for framing such descriptions, evidence of which is retained as an aid to the reader: as we follow certain specific links between each aspect, we will indicate each change of focus with a kind of mock-ambient cue, the prefaces **A:**, **I:**, **T:**, or a combination of the three, for Artistic, Interactional and Technological aspects respectively.

THE PERPETUAL EMOTIONS PROJECT

[NN/MM] *Séa.nce* is part of an ongoing Internet project by Norie Neumark and Maria Miranda titled *The Perpetual Emotions Project*, which requires some discussion here as the locus and context for *Séa.nce*. *The Perpetual Emotions Project* began with a fascination with the motion, rather than sentimental, side of emotion, hence the term 'e.motion'. Emotion in this sense may be understood as feelings that *move* bodies. In making *The Perpetual Emotions Project*, Miranda and Neumark were also interested in blurring boundaries, such as between metaphor/literal, science/fiction, or rational/irrational. They do this by making a 'fictive' work, which establishes a research institute, *The Institute for the Study of Perpetual E.motions* (Neumark & Miranda, 2003) (hereafter referred to as the *Institute*). The *Institute*, under the direction of Doktor Rumor (a.k.a. Norie Neumark) and Professore Rumore (a.k.a. Maria Miranda)—renowned Australian rumourologists—is presented as the leading international centre for the science of emotionography.

Another inspiration for the approach to their work has been the 'pataphysics of Alfred Jarry (the single apostrophe is intentional). In Exploits and Opinions of Doctor Faustroll, 'Pataphysician: A Neo-Scientific Novel Jarry defines his 'pataphysics as: … the science of imaginary solutions and … above all, the science of the particular, despite the common opinion that the only science is that of the general. 'Pataphysics will examine the laws governing exceptions, and will explain the universe supplementary to this one. (Jarry, 1980, p. 192)

THE ORIGINS OF *SÉA.NCE*

Séa.nce arose as an initiative of the *Institute* which explores the artistic and 'pataphysical potential offered by e.motionography in the context of a network of people participating in a séance. The focus of this exploration is on the e.motionographic products of collective e.motion. In keeping with the Internet residence of the Institute, the séance takes place in a networked environment, in which the audience is not necessarily physically co-located. [GT] **A/I:** Through a system which relays the collectivised motion of the players' avatars (a single avatar appears near 'C' in Figure 2), the planchette (the ouija board's pointer or puck—

the shape over 'Q' in Figure 2) is moved around the board from letter to letter. When the planchette lands on a letter it responds with the sounds of the spoken alphabets selected, according to the e.motionographic results of the player's e.motions, from over twenty languages. This selection 'audiolises' the networked emotions of the players. At the same time players type their interpretations, comments, feelings, thoughts and ideas into the message box creating a corresponding textual cacophony.

Figure 2. The Séa.nce interface during development.

[NN/MM] While playful, Séa.nce is not strictly gameplay. Artistically, it is an exploration into issues of non-control and non-instrumentality. Our aim is not to have a 'game' environment with rigid rules and control, but rather to have people play in an environment where they become part of the event. In a way, players are in a networked space which is both controlled and uncontrollable; both individual and collective. These coexisting oppositions are suggested by the term 'uncanny', and our task as designers and performers is to help people to get into that space. [GT] **T:** From the early discussion (the two 'real life' meetings), it became clear that, since the involvement of players over the Internet was a key component of the performance, we were limited by what hardware (and to a lesser extent software) might be available to remotely-located players, specifically by the use of mouse/trackpad (or unconventional use of keyboard) to convey e.motions. We used Macromedia Flash to implement the interface (see Figure 2 for an image of the interface during development), the rationale for which we will explore in the discussion towards the end. To arbitrate the network communication, we commissioned a specially built communications server running in PHP, which was much cheaper to develop and more flexible than Macromedia's own Flash Communication Server (more of this also

in the later discussion). The software-side analysis and performance controls are located in specially-built versions of the interface, for the performers to use. We used off-the-shelf software to manage the video streaming from the live performance.

A/I: The artistic vision of *Séa.nce* presents the artist and interaction designer with several important questions: How can we create an interactive environment where local and remote players communicate in a séance mode? What would it look like, sound like and feel like to produce motion and sound from e.motion that is relaying in the network? More specifically, how do we work with the kinetic motion of e.motion to move the planchette and produce the sound? Our approach to addressing these questions is covered in the next two sections.

ENGENDERING NETWORKED E.MOTION

The production and measurement of networked e.motion requires an interface which encourages collective action and which is sensitive enough to measure it. From the interaction design perspective, several techniques were used to encourage such interaction, and a description of these follows:[NN/MM] Firstly, we extend the system beyond the computer and as far as possible into the physical environment, in order to create a suitable atmosphere in which Doktor Rumor and Professore Rumore can lead the séance. At the physical location of the performance, we hold the event at midnight with dimmed lighting, burn incense and so on, and ask remote networked participants to do the same where appropriate. The interface takes over the computer, minimising the potential for distraction from other processes. To complete the effect, interface elements are brought out into the real world, with fortune cookies containing guidance and advice.

Secondly, an important part of the séance interface was that it should not work via conventional controlled interactivity and should indeed trouble such interaction. This is a difficult balancing act to achieve, because too much enforced blurring of controlled perceptions and actions within the interface may trigger feelings of frustration or anxiety and a rejection of the process, rather like trying to hypnotise an unwilling subject. Although it is not possible to get all audience members to enter the fictive space and interact in a new way (without pre-selecting for suitability, which is something we are not ready to do at this stage), the interaction techniques we employed were designed to assist those who wished to do so. Foremost amongst these was to make the avatars all look and behave the same—what starts as a mildly humorous surprise after login soon becomes an important property of the collective interaction as the conventional player/avatar relationship is troubled and the boundaries of individual identity are blurred in the interface.

[GT] **I/T:** In order to reduce the perceptual impact of making sudden movements, we smoothed out the position data for each player's avatar, interpolating between each avatar's current position and its destination as indicated by the true motion. **T:** This approach additionally addressed the problem that, due to the way the avatars' messages propagate over the network means that to display the raw position data would result in a jerky updating of positions and it would be easy for a player to distinguish his or her avatar from others by the others' lack of flowing movement. To combat this, we smoothed out the position data for each network avatar in the same way. (It is worth noting that the smoothing of motion does not affect the e.motionographic analysis, which happens on unaltered data, so that sudden movements are appropriately analysed, but simply not rewarded in the interface.)

I: The avatar behaviour was rounded off by stipulating that no clicking of the mouse or trackpad buttons should ever be needed during the séance phase (in fact, no clicking is required at any stage after login, except for information request buttons).

[NN/MM] In an effort to diffuse frustration built up by lack of individual control, players are tasked with a series of warm-up exercises, designed to get the player used to the way the interface works, to bond collectively with other players, and to relax and "go with the flow". Figure 3 shows some examples, and Figure 4 shows players carrying out one of the exercises.

Visualization Exercise
Please hold up your index finger at a 45° angle. For participants online please point towards the Baltic Sea. Close your eyes. Visualise the emotional power of your avatar. Picture it moving around the board giving you the answers that you seek.

Breathing Exercise
Next is the Remembering to Breathe exercise. Breathe in slowly through the nose, then out slowly through the mouth while trying to keep your mind blank. Repeat this 3 times. The goal is to get calm and in touch with your emotions. This will help you accept that there is no individual control in the network.

Wiggle exercise
Please wiggle your avatar and then practice approaching the planchette but not touching it. The Planchette is the small green ovaloid shape in the middle of the larger black oval Board. It's looking at you. This will help you get in touch with your avatar. If at some time you lose connection, you can repeat this wiggling.

Noses Exercise
Touch the nose of a nearby avatar. Feel the emotional relay. You may feel more intensity with some players than others. Don't worry, this is considered normal. This will also help you understand the networked e.motions of and through your avatar.

Figure 3. Examples of warm-up exercises

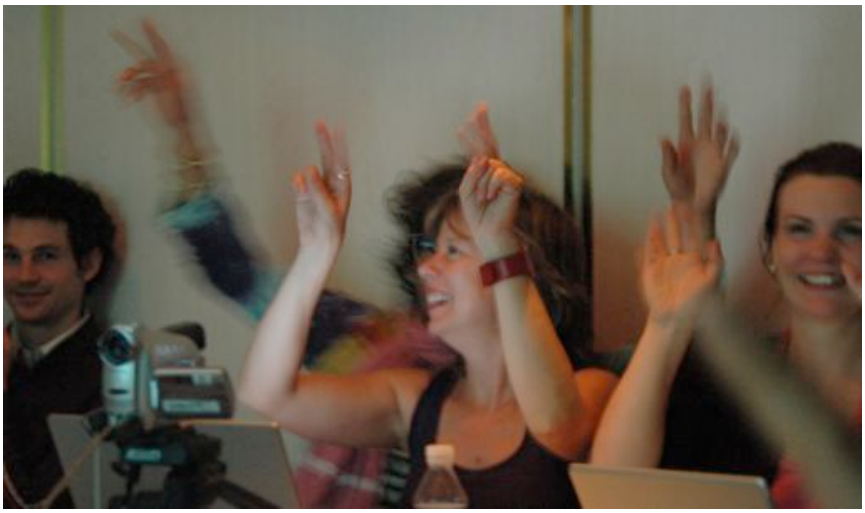


Figure 4. Players warming up for the séance.

[GT] Another important device to compensate for lack of individual control was to engage players in collective dialogue through the message box. This was a place to replace individual game type control with collective textual play. Here are some reactions from the players to their avatars during the first performance at ISEA2004. The excerpts provide an example of how this worked both as a way to 'discuss' the uncanny interface as well as to create it through their engagement (the numbers identify the player):

8	My avatar is an arrow. Will it fade?
14	which one is me?
5	that's funny—my avatar is an arrow too!!
8	Is everyone's avatar an arrow?
…	
14	my arrow is cooler than yours
11	I am not here
…	
8	My arrows are very elegant
…	
10	Cool! I found my arrow, but I propably lost it again :-) [sic]
…	
5	my arrow is calm
3	mine is pointy
4	can I have my avatar back
6	my arrow points in wonder

We are pleased to note comments indicating some success in our efforts: players appeared to equate the avatar with themselves ("which one is me?") and with emotions ("calm"; "in wonder"), and exhibited a disturbed sense of individuality ("I am not here"; "can I have my avatar back"). This suggests a successful troubling of the relationship with the avatar too. [NN/MM] The intense engagement with the message box

also indicates the success of the strategy of involving interactors in making the fictive rather than just playing out a pregiven fiction. It is worth noting that this sense of the potential of the message box developed during the process of collaborating on the work, as discussed below.

PERCEIVING NETWORKED E.MOTION

[GT] Having provided the environment in which to stimulate the production of e.motion, we were faced with several tasks. First, we had to find ways of detecting e.motion using the limited hardware we had at our disposal. Then we had to transform the collection of data so it could be analysed, both by software (in determining the movements of the planchette and the sounds to be played), and by the players themselves in their interpretation of the planchette's movement and the sounds.

The inseparable production/detection/analysis (or 'perception') of e.motional messages is constituted within a cycle of stimuli (instructions and performance events) from the performers, and counter-stimuli (question-asking, planchette moving, answer-receiving, interpretation) from the players and the software itself. [NN/MM] After the warm-up exercises, a "spirit guide" is introduced, who acts as a commentator on the proceedings and a mediator of the questions asked to the board. The players are asked by Doktor Rumor if they have any questions for the board, and can type in suggestions. The Doktor selects one of these questions and the e.motional relay begins.

[GT] **A/I:** *Séa.nce's* input is made to be as sensitive as possible, given the technological limitations imposed by the mouse or trackpad and conventional operating system handling of pointing device (for instance, it is not generally possible to move the mouse beyond the screen boundaries) so that the system can elicit the maximum amount of data about the players' networked e.motion. The data needs to be aggregated and processed in order to be usefully analysed, both positively by the séance planchette (by 'positively' we mean analysis of that which is posited by the data), and interpretively by the players themselves (by 'interpretive' we mean a culturally-derived analysis of the data). An interesting way of looking at how this aggregation has come about in *Séa.nce* is by looking at it as a 'pataphysical effect of incorporating the 'séance' paradigm into an 'interactive system' paradigm, and the prominence of the term 'medium' in each of these. The séance-derived meaning of 'medium' (with plural 'mediums') describes a clairvoyant (literally 'clear-seeing'), who is thought to have the power to communicate with the spirits of the dead or with agents of another dimension. The second meaning, more usual in interactive systems (with plural 'media') is a means of communication, or a framework through which something else is conveyed.

In the way that we have combined them, these media/mediums become not so distinct. Broadly speaking, we could say that the culture-medium in *Séa.nce* can be found in the use of the body and physical interface devices of the players to make and interpret gestures, whereas the clairvoyant-medium is found in the software's own collection/collation and interpretation of these gestures. However, by mutually interpreting, and thus influencing, each other, these 'medium' processes oscillate and resonate with each other to give rise not only to amplification, which is how e.motionographic representations can become apparent to recognise, but also to feedback loops, which can influence the construction of new meanings of *Séa.nce* results by the audience—medium becomes glossalalia.

T/A: The manner in which this interplay between, and consequent aggregation of, analyses and influences takes place is the most difficult aspect of *Séa.nce* to get right. Specifically, we spent most time on the way in which the avatar data are aggregated and how both the planchette and the subsequent sound react uncannily to that aggregation, and on modifying the performance to better inform the players about the space in which the interplay takes place.

Numerous techniques suggested themselves: we started with a realistic physical model of a séance, where the avatars represent force vectors, and the planchette (the pointer of the ouija board, remember) moves and rotates on a frictional surface according to the sum of these forces. The idea was that by summing several small forces we could produce an uncannily large force, in much the same way as a physical ouija board is (according to the sceptics!) supposed to work. Initial user testing showed, however, that this model was very difficult to control without hours of practice, because the haptic control and feedback afforded humans in a real-life séance is sorely lacking in this online simulation. Had the players used a physical ouija board with electronic position sensing (or even haptic devices over the network) as input, this model might have worked marvellously.

This was initially replaced by a simple averaging algorithm. The planchette moved to the average position of nearby avatars. This was much easier to control, but hence completely lost any feeling of uncanniness—the planchette was manifestly following the avatars, not the other way round.

The next method we tried, and the beginnings of the one that *Séa.nce* currently uses, was to have the planchette point in the average direction of nearby avatars (which was later refined to having the planchette point in the average of the directions indicated by the letters which the avatars pointed to). This prompted a redesign of the board to be ovular (it was originally rectangular), so that the planchette had a more evenly-distributed range of letters to which it could next move. In further in-

formal tests, this was found to sometimes produce the uncanny effect (later described as the "fun" movements by one of the testers). Quite often, these movements were inhibited by avatars which hadn't moved since the last planchette decision, so were guiding the planchette to the same or a nearby letter. We addressed this by introducing "e.motional energy" for each avatar—the less an avatar is moved, the lower its energy becomes, and the lower its influence on the planchette (this is why the avatars go transparent when left still). **I/T:** Having settled upon the fundamental modus operandi for the planchette, the remaining refinements consisted mainly of determining precisely under what e.motionographic circumstances this uncanny movement should be produced. The main factors are: the proportion of avatars that need to be influencing the planchette, how much their direction agreed, what their e.motional energy should be, how that compared with the energy of the non-influencing avatars, and what counts as influence anyway. These factors, derivatives and others also inform the generation of the sound for the planchette's decision. The sound is composed of the nth letter (n is indicated by an animation of a number near the letter) of each of a selection of recorded alphabets from different languages, played at different times, rates and auditory positions, and is distributed amongst players' computers (except for remotely-connected players, who receive a stereo equivalent). The performance script was modified to support the behaviour of these factors.

The way the factors behaved and the nature of the performance were significantly informed by our user reports. The most interesting example of a significant change arose from a confluence of factors that became clear through the user testing and feedback. Firstly, a problem with the planchette movement meant that it tended to get stuck between two letters. This disrupted the 'uncanny' feeling and meant that participants were encouraged to make conscious actions: "I enjoyed that it seemed to have its own life… it made me think I wasn't supposed to play an active role in the decision making… but then when the words seemed to come out as neighbouring letters, I thought it would be more interesting to send it onto the other side of the board or try and actually make a word, and then I wanted to influence it and didn't feel that I could."—Trial participant.

Secondly, both performers and players found that, by the time the board had given its answer, it was hard to remember the question. Thirdly, some participants commented that the visual display of the answer, the literal letters, was not as rich or ripe for interpretation as the sound. To deal with all of these, we decided that (as well as fixing the problem that got the planchette stuck), we modified the control panel to allow highlighting of a question, which was then displayed in the box below the board, which had hitherto been used to show the answers.

I/A: After each question–séance–answer-interpretation pattern, the cycle is repeated until the end of the performance.

NETWORKED CREATIVE COLLABORATION—INSIDE AND OUTSIDE OF SÉA.NCE

[GT/AW/NN/MM] It was a new experience for all parties to be involved in an entirely online collaborative development—the total face-to-face meeting time before installation on the final hardware was 2 hours of preliminary meetings, followed by 12 hours of work before the first performance at ISEA2004 (where co-location is mostly a necessity). One might think that we relied on the next-best thing to face-to-face interaction, such as videoconferencing or telephone calls, but we found these completely unnecessary except for at the performance trial. We used email, FTP (for transferring files) and *Séa.nce* itself as our sole means of communication. (One exception: we used iChat once, before the messaging function of *Séa.nce* was finished).

However, we do not feel that the development process was stunted by this lack of face-to-face interaction. On the contrary, the very nature of the medium we are working with (namely networked interactive systems) means that we can collaborate through manipulating that medium, much as sculptors may collaborate through manipulating clay (or from the technologist's perspective, much as open-source programmers may collaborate through manipulating code). We found that using the medium as the medium for collaboration on the medium calls for a certain discipline, but also that we quickly evolved a certain shared tacit knowledge—just by studying each others' interactions with the embryonic piece we were able to get a sense of each others' concerns with it and thus react accordingly. This tacit awareness could not have been gained from interacting via another medium.

We realised that not only are we collaborating through the medium we are shaping, but also that the collaborative process itself shapes of the medium. So we are able to exploit the conveyance of tacit knowledge in the *Séa.nce* performance—we have designed the interface so that players are able to gauge the other players' involvements with the interface and react accordingly, in much the same way as we collaborators have done during development. As mentioned above, one place this happened for us was as we used the message box to communicate while séa.ncing, as a form of collaboration, and, in so doing, realised its rich artistic and interactive potential.

A short word on the role of email in our collaboration: email fulfils one particularly useful function that *Séa.nce* as a work hasn't needed to—off-line working. So, for example, collaborators in Sydney could, one evening, email a set of questions and thoughts to collaborators in Paris, and arrive the next morning to find responses and updates, and vice versa.

This, combined with the several shared waking hours that the time-difference affords us which can be used for higher-frequency-yet-still-no-need-for-realtime communication, meant that there was pleasantly little waiting around for the other party to formulate a response.[Here I have omitted a brief discussion of this work in relation to my thesis, which appears in expanded form after the paper extract]

CONCLUSIONS

[GT] The anthropologist Lucy Suchman, in a commentary on affective interfaces (Suchman, 2002), contrasts two main types of emotional machine, those which attempt to simulate human emotion (which, argues Suchman, is a "fetishised humanness, stripped of its contingency, locatedness, historicity and particular embodiment") and those which evoke human emotion (an artwork she gives as an example is described as "an emblematically human encounter"). She concludes:"…affective encounters at the computer interface are those moments of moving complicity between persons and things achieved through particular, dynamic materialities and extended socialities."

Such e.motional encounters can be understood as arising between bodies and machines (Neumark, 2001). It became important when designing the system to work with this sense of e.motions that are not entities but motions that relay between actors. A phenomenologists might say that the setting into motion of, and perception of, e.motions is active, embodied and always generative of meaning—one cannot detect e.motions without at the same time analysing them, and conversely, e.motions are meaningless until they are detected. It is interesting to note here that this way of thinking about the motion of emotions also plays with and interrogates the compulsive twitch behaviour of some forms of gameplay and interactive behaviour.

The interactive artist David Rokeby investigates the social responsibilities carried by the creators of interactive systems (Rokeby, 1998), by looking at the long- and short-term effects that features of interfaces have on users' perceptions of the world. He writes:

> *The process of designing an interaction should also itself be interactive*
>
> *[…] we need to expand the terms of [an evaluative] feedback loop*
>
> *… to include an awareness of the impressions an interaction leaves*
>
> *on the user*

This describes well the approach we have taken in designing Séa.nce, but we have extended the concept to apply to collaborative design of collaborative interactions. To paraphrase Rokeby, the process of designing a collaborative interaction should also itself be collaborative … we

need to expand the terms of [an evaluative] feedback loop … to include an awareness of the impressions an collaboration leaves on the user.

—End of paper extract—

SÉA.NCE IN THE CONTEXT OF THIS THESIS

Séa.nce is an example of an interactive art development process that quickly came up against the limitations of its development environment—Macromedia Flash. As the technologist in the collaboration, my rationale for the use of Flash was a combination of availability of the necessary tools, and ease of distribution. Flash creates high-quality vector graphics in realtime—its historic strength, and mostly what is required for this application. Additionally, it publishes to a single cross-platform file, the .swf, which contains all of the information for any Flash player to be able to run Séa.nce, which is useful for people participating over the Internet.

Flash's programming language is called ActionScript. Now, ActionScript is a fully object-oriented language, which is not dissimilar in difficulty to other object-oriented languages. Given time and familiarity with the API, someone who could program *Séa.nce* in ActionScript could program it in another, less limited language, such as Objective C. Flash simply builds in certain, unchangeable, graphics and interaction tools and hooks, which while initially helpful, are the kind of rigid, yet technologically arbitrary limitations discussed in Study 1 (Chapter 5). Although the language per se was not a problem for me, I repeatedly found myself confronting the built-in limitations of the Flash system, and had to devise ingenious workarounds, or work manually, to compensate, rather than being able to modify the way the system worked. These limitations were:

Webcam Video: The version of Flash current at the time of Séa.nce's development could play back video from files, but could not stream live video. The workaround was to post periodically post live images onto a server, and have each client download those images. The speech from the performance was transcribed and broadcast as text messages (since a lot was scripted, we could build an interface to assist this).

Batch Processing: Each letter sound in Séa.nce is a separate sound file, and there were approximately 250 of them. Each sound had to be imported into Flash, named, made dynamic and inserted into the preload movie by mouse click. There were no keyboard commands, and no programming hooks.

Mouse position: We originally wanted to redefine the behaviour of the mouse, having the mouse cursors moved 'supernaturally' by the computer, but Flash only lets us read (not write to) the X/Y coordinate of

the cursor, which is limited by the constraints of the screen. We hid the actual cursor and replaced it with an avatar, but if the avatar is moved without the mouse cursor, the two get out of sync and the user cannot move the avatar everywhere around the screen. We eventually had to abandon the idea of cursors moving without their users.

Graphics: Flash's way of drawing anti-aliased vector shapes is its signature, both functionally and aesthetically. While impressive in its early surroundings of aliased, static, bitmapped graphics, the look has now become cliché. To an extent, I could work around this by dirtying the interface with static noise and shakes [screenshot], but the aesthetic stamp of Flash on Séa.nce is evident.

It is doubtful whether we will see Flash, in the form that we know it, develop the kind of flexibility we needed, or that I envisage in a general environment for building interactive art. Such a system would have to be created from the ground up, in itself, to allow the user to 'drill' back down as far as he or she wishes to make modifications. *Séa.nce* could conceivably have been written in Squeak, but Squeak's conceptions of ease of use and understanding (and aesthetics) still leave much to be desired (see Appendix 2 on CD), making it difficult for a non-technical artist to engage with the environment. Although its cross-platform distribution mechanism is at least on a par with that of Flash, the Squeak environment presents a different kind of barrier to the non-technical user trying to run it.

Summary

cubeLife was my first interactive art piece, and I was persuaded by the requirements of my course to develop it in a traditional software engineering process. This process was of mixed success, but the artwork itself awoke my interest in the human, creative side of computing, rather than the technological, engineering side that had been the emphasis of my education so far. The development also highlighted a need to more easily develop exploratory interfaces in Java.

Séa.nce was a more recent collaboration, designed as a networked performance and written in Flash. *Séa.nce* is an example of an interactive art development process that quickly came up against the limitations of its development environment, because Flash builds in certain, unchangeable, graphics and interaction tools and hooks. I had to devise ingenious workarounds, or work manually, to compensate, rather than being able to modify the way the system worked.

My descriptions of these pieces will be echoed in the chapters to come, as we study technologists who have come up against similar limitations and required similar features of technology. In the next chapter I will make an argument that these limitations are not fundamental to com-

puting, so that we can remove them in designs for future computing systems.

5. Study 1—The Role of Computing Media in Interactive Art

Introduction

Before we study interactive art collaborations, or make recommendations for useful technology and methodology to support interactive art, we need to figure out what we're missing out on that useful technology and methodology should support. In other words, what is the creativity support potential of the computing medium? What is preventing us from exploiting that full potential?In this chapter I argue that the creativity support potential of the computing medium is great for many reasons, but crucially because its flexibility means that computing sub-

media (applications, tools and commands) can, in theory, be changed in order to match a creative user's changing conceptual spaces.

However, the reality we see is quite different. Rather than being allowed to change software in ways we can envisage, we are prevented from doing so because software is compiled. My argument in this chapter will be that there is nothing fundamental about computing which means that the compiler must prevent us from changing software, because sufficient counterexamples can be found to show that this cannot be the case.

Terminology

Before I begin the argument, here are definitions of terms that I use:

A **computer** is a machine for manipulating data according to a list of instructions. This manipulation is called **computation**.

Computing is *human* activity that involves computation. This definition, rather unconventionally, emphasises the human perception of computation.

A **medium** (plural *media*) can be described, according to Marshall McLuhan, as any extension of man (McLuhan, 1964). According to the philosophy of Heidegger (in particular) (particularly Heidegger, 1962), tools, when used, are extensions of man, and so any use of a tool counts as a medium. Computing—using computers as tools—is an extension of man, and is therefore also a medium.

Duchamp's *Fountain*

Every medium has **constraints**. The constraints of a medium are a combination of fundamental limits and socially-defined norms. Examples of fundamental limits are gravity, electricity, mathematical axioms, and so on, which are generally considered to be impossible to change or overcome. Socially defined limits are what society considers a medium to be, or not to be, and can be prone to change or to be overcome. For example, society may consider 'sculpture' to be a shape carved out of stone or modelled in clay by an artist, but not a pre-existing object merely shown by an artist. Of course, Marcel Duchamp turned such a definition of sculpture on its head by presenting 'Fountain' (1917), a 'ready-made' sculpture consisting of a ceramic urinal. Duchamp's exam-

ple shows that constraints of media can be *overcome* (by works which don't conform to accepted constraints) and *changed* (by works which are powerful enough to cause a social reconsideration of the constraints of a medium). 'Fountain' may not have been a sculpture (in other words, it overcame prevailing social constraints), or it may have caused the medium of sculpture to be redefined (thus it changed those constraints).

From the notions of medium, and constraints of a medium, we can introduce the term **sub-medium**. Sub-media impose tighter restrictions on activity than their super-media. This is because they share all the constraints of their parent media, and more besides. For example, oil-painting is a sub-medium of painting; Microsoft Word is a sub-medium of computing. In computing, sub-media can be complex applications or programming languages, or simple commands or tools.

Computing as a Creative Medium

As I outlined in the introduction to this thesis, we are concerned with computing in particular because it is a flexible and unique medium, and after only fifty years, still full of untapped potential for creativity support. To elaborate: when we are creative, our conceptual spaces change (as Boden explains, see the literature review, p. 23). For a medium to support creativity, it must adapt to a creative person's changing conceptual space, which means it (or its super-media) must have flexible constraints. In fact, the history of creativity is full of examples where these constraints have been overcome or changed. For example, the adoption of oil paints (which dry more slowly than previous paints) meant that painters could take more time to finish a painting.

Computers are flexible in four particular ways, which I call the four Ss. These are **speed**, **servitude**, **synaesthesia** and **structure**. These qualities are modulated and exploited in different ways by different sub-media.

Speed is, simply enough, the quality of a computer which enables it do certain things quicker than we can. We use computers to try things that would otherwise require too much time. This quality was the impetus for the initial development of computers, from Babbage's difference engine to World War II, to calculate missile trajectories and solve encryption problems which could not have been completed quickly enough

by hand. In interactive art, as well as in a great many other computing systems, the goal is often to generate the response 'in real time'.

Servitude is descriptive of both the incredible cheapness and unquestioning obedience of computation, and it is what allows artists and programmers to create massive and wide-ranging programmatic edifices—doing something a million times is hardly more difficult than doing it once. John Maeda has a nice analogy:

> *When preparing to embark on a dangerous rescue mission in the now cult film "The Matrix", the main character Neo says, "We're going to need guns. Lots of them." The moment these words are uttered, cases of weapons speed in from the distance like bullet trains into the null zone that Neo and his colleague occupy… I recall this scene… for the sense of magic that occurs when Neo expresses his wish. The instantaneous rush of tremendous resources… epitomises for me the experience of freedom when programming the invisible spaces of computer codes.*
>
> (Maeda, 2004, p. 17)

These two of the four Ss, speed and servitude, are early, and perhaps the most obvious qualities of computing. In a nutshell, servitude is being able to say "light a million pixels" and have it obeyed, no matter what. Speed is being able to say "light a million pixels" and to have it happen in 10 milliseconds.

Synaesthesia, outside of computing, is a neurological condition in which a person experiences sensations in one 'sense', when another 'sense' is stimulated. For example, a synaesthete may 'see' colours in response to musical sounds (Ramachandran & Hubbard, 2001a). As we learnt in the literature review (p. 34), synaesthesia appears to be more common in artists, poets and novelists than in the population at large (Ramachandran & Hubbard, 2001b), which Ramachandran and Hubbard explain by an increased 'cross-wiring' of the regions of the brain that deal with these different senses. I am using the term synaesthesia within computing as an analogy to describe Negroponte's 'formless bits' (Negroponte, 1995)—that all a computer does is perform operations on collections of 1s and 0s.

This means it is necessary to represent everything else—data and instructions—as these 1s and 0s. Inside a computer, video is the same stuff as audio is the same stuff as text is the same stuff as commands is the same stuff as time, which means it is possible, for example, to combine them and convert between them computationally, and to present the combined or converted experience to a (less-synaesthetic) audience. Synaesthesia is itself strongly exploited in interactive art, as a consequence of this quality, and the effect can be very powerful. The artist David Rokeby writes:

> *In this work [Very Nervous System], I use video cameras, an artificial perception system, computer, and synthesizer to create a space in which body movements are translated into sound or music in real-time. An hour of the continuous, direct feedback in this system strongly reinforces a sense of connection with the surrounding environment. Walking down the street afterwards, I feel connected to all things. The sound of a passing car splashing through a puddle seems to be directly related to my movements. I feel implicated in every action around me. On the other hand, if I put on a CD, I quickly feel cheated that the music does not change with my actions. … Interfaces leave imprints on our perceptual system which me carry out into the world.*
>
> (Rokeby, 1998)

Structure is perhaps the quintessential, and most distinctive quality of computing. Whereas we can experience speed, servitude, and synaesthesia elsewhere, computing's structure, in its most abstract sense, is unique to itself. The closest analogy we can make is to our own minds (the analogy, and its psychological effects on those who use computers is described by Turkle (1984)).

Turing's proof that an infinite Turing Machine could calculate anything which could be calculated (Turing, 1937), and von Neumann's implementation of a finite Turing Machine in computing architecture (von Neumann, 1945), set out a basic structure of computing as a medium with potentially highly-changeable constraints. However, as a consequence of its mathematical underpinnings, computing's structure is

highly abstract, consisting of arrangements of 1s and 0s and simple operations upon them. In order to communicate in other ways through this pliable yet abstract binary medium, we have to learn how to represent the concepts we wish to communicate using this medium. This means that we need to create more concrete sub-media within this abstract binary framework. On the other hand, we need to make abstractions about problem-domain situations that can be manipulated by computation.

Abstracting situations into structures, and concreting binary into those same structures, the con*struct*ion of computing's metaphors, if you will, is the current distinctive speciality of programmers and systems analysts. In interactive art, artists often employ programmers and systems analysts, because, though they may be confident of the capability of a computer to realise their creative imaginations (by running an artwork's software) they lack the programming expertise to translate what they imagine into program code.

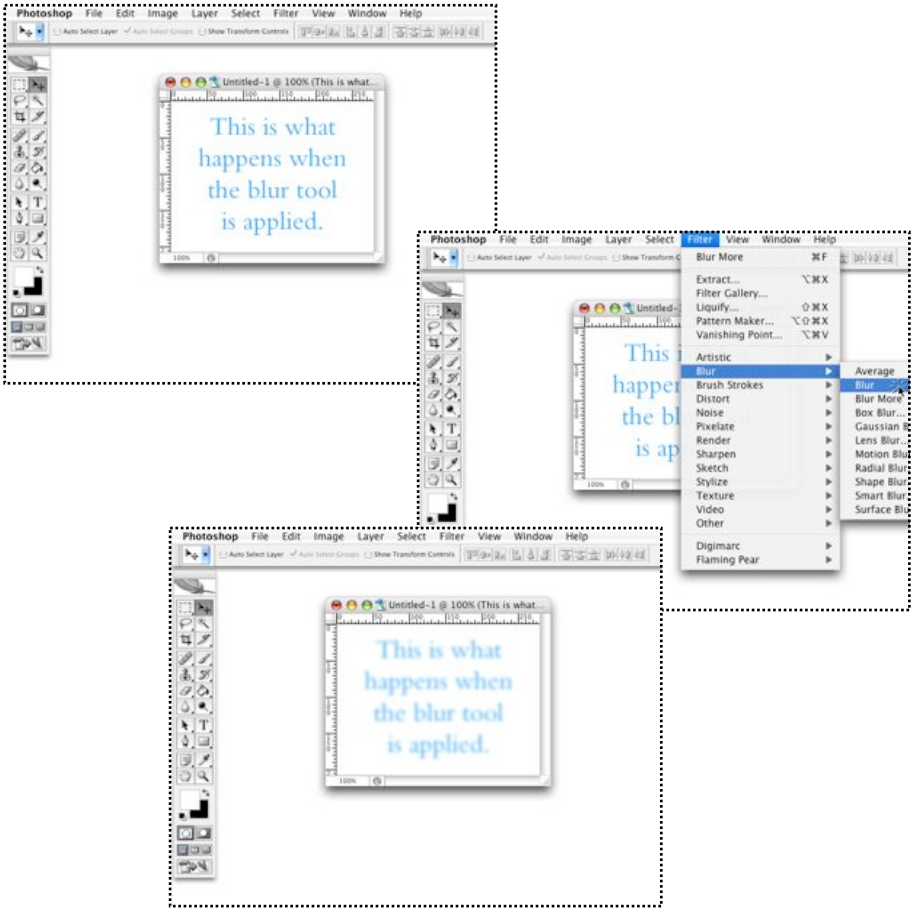
These four Ss, the speed, servitude, synaesthesia and structure of computing are keys to the medium's support of creative flexibility. The medium supports sub-media (software tools) flexible enough to represent a wide variety of real-world concepts as structures of binary numbers, and for changes to the structure or meaning of the concepts to happen relatively quickly and cheaply. So given that the computing medium fundamentally supports flexibility, how, and to what extent do its sub-media also support, or constrain it?

Rephrasing the question, ideally we need computing sub-media—software tools—to be flexible enough to move constraints with our conceptual spaces. In theory, as described above, the computing super-medium supports such flexibility. In practice, we commonly experience software tools to be inflexible and unchangeable, which restricts our creativity. To an extent programmers can exploit computing's flexibility more than non-programmers, but even programmers too can come up against constraints they cannot change. What gives rise to these constraints? Are they a necessity in computing—is there something fundamental about computing which, despite the medium's flexibility, causes its sub-media to be constrained in the way we experience? In the

next section I shall present an example of constraints around sub-media that restrict creativity, in order to show what can cause them.

Examples of Constraining Sub-Media

Adobe's Photoshop software is a sub-medium of computing. Photoshop is a popular piece of software used by those in creative industries. It has even inspired the interactive artwork Auto-Illustrator[1] , a kind of 'spoof' of Photoshop, which, in its own way, pokes fun at the very constraints that I here discuss.Photoshop's 'Blur' tool is a sub-medium of Photoshop (and therefore also of computing). This is what happens when the Blur tool is applied.

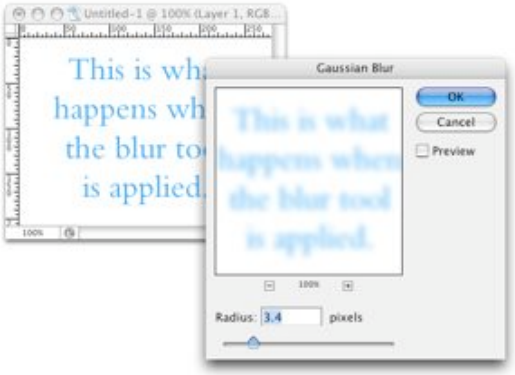


A simple Blur operation in Photoshop

There are no options or parameters to adjust for the Blur tool—the algorithm, whatever it may be, is executed straight away on the entire im-

[1] http://www.auto-illustrator.com/

age selection. The constraints of the Blur tool are that it is either applied to the selection or it is not. Photoshop has added other Blur tools over the years, in response to the limitations of existing ones, some with more parameters than others (see the Gaussian Blur example below), but the algorithms which make them work are similarly hidden, and so a similar argument applies. The inflexibility of Photoshop's blur algorithms in part lead to the Photoshop 'look' (Photoshop's equivalent to the Flash 'look', mentioned in Chapter 4), in which the aesthetic result of common algorithms is so recognisable as to be almost clichéd, and which creative people must work hard to subvert.



The Gaussian Blur interface

Now, imagine that a creative user imagined a way to blur an image in a new fashion—by choosing the amount of blur in each colour channel, for example[2] . As he or she imagines the change, the creative user changes his or her conceptual space, and requires a tool, a sub-medium with different constraints, to have to hand support this. How might this required sub-medium come to hand?

The simplest way for a required sub-medium to come to hand is if the sub-medium exists already, and the creative user is aware of it and has the means to acquire it. For instance, a creative user may imagine a blur with a varying radius of pixels, in a Gaussian distribution. This sub-medium already exists in Photoshop as the Gaussian Blur tool, so there would be little point in re-inventing something that does the same job. Although other tools within Photoshop are close to hand, there may exist appropriate tools in other places that the user may not be aware

2 This is possible to do in several stages in Photoshop, but no single, simple exploratory tool exists

of—supporting the discovery of those tools is a matter for further research.

Assuming that a required sub-medium does not already exist (to the user's knowledge), it must be created. This creation could happen *from scratch*—in our example, a new application that blurs differently in each colour channel—or by *changing* an existing tool, if possible.

We are used to the first case—fulfilling a requirement for a new sub-medium is the impetus for almost all conventional software development, and so requires conventional programming expertise. However, the result is not that the constraints of our tools have changed to support our changing conceptual spaces, but that we have discarded previous tools and made new ones from scratch. It seems that true creativity support could be more about changing existing tools than about rebuilding them from scratch.

So let us explore the scenario where an existing tool is changed. In our example, if the change is an easily-imagined change to the Blur tool—a change of a few characters of code, say—how might the Blur tool in fact be changed? Of course, the way to change a tool is to convince someone who is capable of making the change that it is worth doing, then wait for them to do it. This approach immediately seems rather passive, but we will explore it in our example. Who is capable of making a change to Photoshop's Blur tool? Currently, only the custodians of Photoshop's source code, Adobe employees, who are guided by the software's designers. So the designers must be convinced that this creative user's imagined change is a good idea, order the change to be made, then the creative user must purchase the next release of Photoshop in order to see the change.

Clearly this is an absurd scenario in terms of creativity support. Photoshop's designers can't implement all of the changes desired by their customers, not least because they would conflict (one person's improvement to a Blur tool may be contrary to another person's), but also the time lag involved is well out of step with an individual's creative process. Nor can the designers design for all of the creative uses to which their software could be put. The software's designers may never

make a change that is only useful for one person, and that person is denied the ability to make the change for him or herself[3].

I've used a simple example of modifying a blur tool, but the variety of examples to which my argument applies is vast. For a start, it's possible to imagine a wide variety of ways to change the blur tool:

- At the very simplest, tweaking any supplied sliders will change it (and the very simplest is all we're afforded).
- The filter could be modified to use a different blur algorithm, as in the example above.
- The filter could be modified to do something to the image other than blurring, or blurring something other than the image.
- The filter could be modified to do something entirely unrelated to its original function, like play a game of Tetris.
- The filter could be modified to modify itself.
- And so on.

…and this is just Photoshop's blur filter. The majority of tools, in the majority of (closed-source) applications have similar limitations—that any modification outside of the envisaged use is denied.

The variety of these changes range from what is considered "using" to what is considered "programming"—but how do we know what should be done by users and what should be done by programmers? There are many definitions of programming which attempt to distinguish between programming and using. To take a consensus example, Wikipedia's definition (at the time of writing), is that programming is "the craft of writing a set of commands or instructions that can later be compiled and/or interpreted and then inherently transformed to an executable that an electronic machine can execute or 'run'"[4] —with the implication that

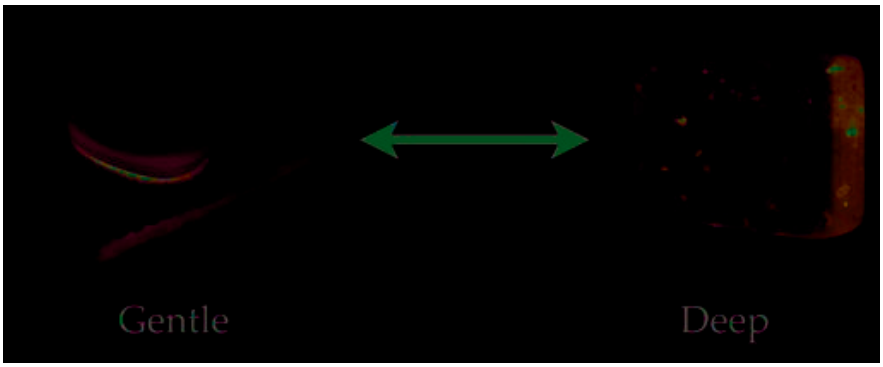
3 Photoshop provides a mechanism for adding to its filter list, but being able to add functionality in a limited fashion is not a replacement for being able to change other functionality

4 http://en.wikipedia.org/wiki/Programming

'using' is un-compiled, or immediate. Other definitions specify that programs be stored for re-use, rather than being immediate and ephemeral, or that they possess control structures and abstract representations, rather than a list of 'literal' commands. These definitions reinforce such separations, creating particular 'mythologies' of fundamental computing constraints. Instead, I want to step beyond the assumption that programming is a separate activity to the rest of computing, and examine what that means for computing as a medium for creativity support.

For a start, it is well-understood that computer interfaces can be viewed in linguistic terms, with mouse movements and key-presses treated as gestures with meaning, and sounds and objects on screen treated as signs with further meaning. Programming, like using, is also a linguistic activity. Conversely, programming languages are no more nor less than a type of application—an application to produce programs. As Wendy Chun says, "Your belief in software itself as something readable and manipulable is an effect of software" (Chun, 2005)—so our belief about software is consequently constrained by that very software.

We can arrange envisaged modifications to computing sub-media on an axis ranging from 'gentle' modifications, to 'deep' modifications (MacLean et al. (MacLean et al., 1990) describe learning these various forms of programming as a 'tailorability mountain'). Gentle modifications are what we make when we 'use' computers, according to conventional understanding of the word; deeper modifications are what we make when we conventionally 'program'. We could say that programming interface representations, say, is 'quite' deep, and programming fundamental computer operations is 'very' deep. The important point is that we can re-characterise 'using' as being gentle programming. I chose these names because neither is pejorative—gentle modifications are as attractive and necessary as deep modifications. This terminology is, coincidentally, shared with cosmetics manufacturers, who produce 'gentle' cleansing and 'deep' cleansing products.



Gentle modifications are skin-deep, easy and quick. Deep modifications get under the skin, and are more difficult and take longer. Gentle modifications would overcome or change few constraints; Deep modifications would overcome or change many constraints.

So changing the blur filter, whether the change is gentle or deep, is fundamentally outside the Photoshop sub-medium as we know it, and thus restricts the creativity of those who would like to make those changes. What are the constraints that situations like this come up against? Are they fundamental constraints of any computing sub-medium?

The constraints that we see here comprise a hierarchical separation between *programming* and *using* a computer. Users 'use' the Blur tool; Programmers 'program' it, and thus possess the power to access its underlying workings through program code. There is no way for users—the very people who require and creatively engage with the tool—to access anything about the blur tool except for its interface and results. Even if the user has enough skill at programming, the blur tool still cannot be changed. The same applies to every closed-source computing system.

The technological explanation for this inability to change software is simple. Computer applications commonly use compiled languages. This means that the program code that a programmer writes is compiled into assembly language (processor commands written as text), and from there is converted into machine code (1s and 0s), which can be run on a particular computer. The conversion process is one-way, so it is generally not possible to recover the original program code from the compiled program. The original program code is generally not supplied to users

(hence the term 'closed-source')[5] .The compiler is the 'wall' that separates programming and using. One cannot use a program until one has compiled its program code; one cannot 'un-compile' a program one is using in order to change its program code. This means that programming and using become *conceptually* separate. Some changes are permitted without recompiling, and some (most, probably) are not, so the compiler wall leads to the common belief that programming is separate from use, and to start programming requires a big 'leap' away from 'merely' using. Yet when envisaged changes are arranged on the gentle-deep programming continuum as above, it is difficult to see which of these changes count as programming, and which do not. A user's perception that "I can't program" is, in that light, a limiting one, and one that arises out of the compiler's limitation.

The compiler wall is also *temporal*, because to make a change to a piece of software in use, a programmer must quit the program, change the program code, save the program code, compile the program code and run ('execute') the program. Whilst it is a quick repetitive process which in itself requires little time or thought, the code-compile-execute cycle requires the program to halt in order to be changed, which means there is a cognitive disconnect between making a change and witnessing the results of that change. The disconnect places strain on working memory, since the behaviour of the (no longer running) program must be remembered during programming.

In most compiled programs, the programming takes place in a programming space—a (text) editing application, which is separate from the running space, where the program runs. The conceptual separation imposed by the compiler means that there is no direct link between running-space and programming-space, so they are separated. So the compiler wall also produces a *spatial* separation between programming and using.

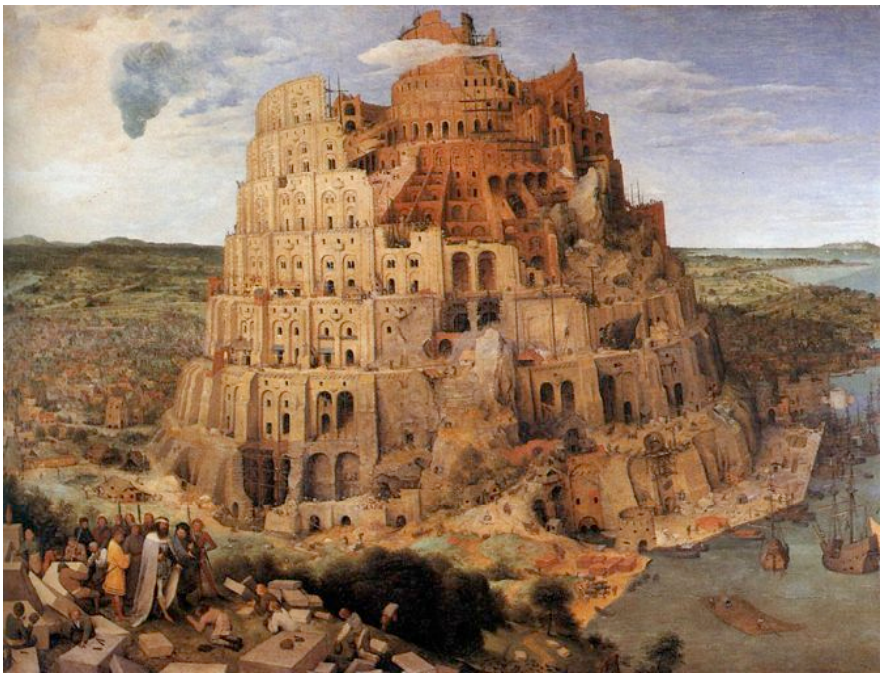
For programmers, compilers put a wall between our description of the hardware operations and the operations themselves. This abstraction is

5 de-compilers exist, which can convert a compiled program into assembly language, but assembly language is fairly incomprehensible in comparison with the program code which produced it

This is not Magritte's painting of a pipe.

what allows the descriptive power that we are used to in programming languages today, but means that programmers lose the ability to "see over the compiler wall". Not being able to see over the compiler wall means programmers can't see precisely what the computer is being asked to do, they can't see 'into' other programming languages in any given language, they can't see into the compiler itself. This distinction is like that between Magritte's painting of a pipe and the pipe itself—a program described in a compiled language denies access to the program *itself*.

The repercussions of the compiler wall are far-reaching. The inability to change compiled software means that restrictive constraints in computing are more prevalent than simply between programming and using. Earlier we treated modifications as lying on a gentle—deep scale, and compiler-caused constraints are everywhere on this scale—on the gentle side, between one user's tool and the next; and on the deep side, between one programming language and the next (compilers are themselves compiled!). The thousands of applications and hundreds of programming languages, each designed separately and compiled separately has given rise to a 'modern Babel'.



The Tower of Babel by Pieter Brueghel the Elder (1563)

> *And the whole earth was of one language, and of one speech. And it came to pass, as they journeyed from the east, that they found a plain in the land of Shinar; and they dwelt there. And they said one to another, Go to, let us make brick, and burn them thoroughly. And they had brick for stone, and slime had they for mortar. And they said, Go to, let us build us a city and a tower, whose top may reach unto heaven; and let us make us a name, lest we be scattered abroad upon the face of the whole earth. And the Lord came down to see the city and the tower, which the children builded. And the Lord said, Behold, the people is one, and they have all one language; and this they begin to do: and now nothing will be restrained from them, which they have imagined to do. Go to, let us go down, and there confound their language, that they may not understand one another's speech. So the Lord scattered them abroad from thence upon the face of all the earth: and they left off to build the city. Therefore is the name of it called Babel (confusion); because the Lord did there confound the language of all the earth: and from thence did the Lord scatter them abroad upon the face of all the earth.*
>
> Genesis 11:1-9

Open interchange formats, such as text, XML and TCP/IP, can ease the situation somewhat, but a) only if the system designers allow their use, and b) this only postpones problems which arise if those formats are limiting for some new creative task. As Marshall McLuhan said, presumably paraphrasing Winston Churchill's comment about buildings, "we shape our tools and thereafter our tools shape us" (McLuhan, 1964).

I asked earlier whether these limitations were fundamental to computing—that is, whether computing works in such a way that compilers *must* conceptually, temporally and spatially separate programming from using. If counterexamples exist—if it is possible to find examples of compilers that don't conceptually, temporally or spatially separate programming from using—then these limitations *must not* be fundamental. It is in fact possible to find such counterexamples, meaning that the di-

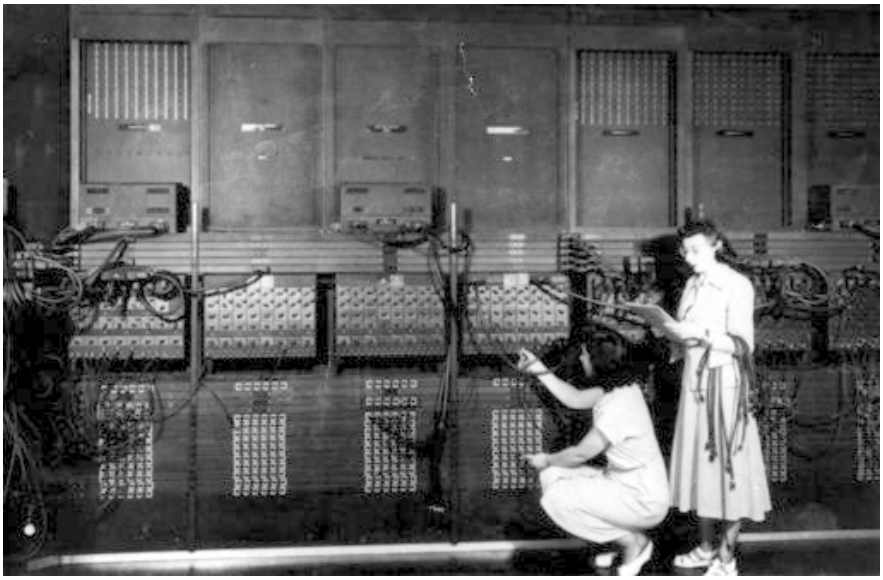
lemma between programming and using is fundamentally false. The next section describes these counterexamples.

Some Counterexamples

The systems described here present counterexamples to the notion that using and programming computers must be conceptually separate. As the last section described, conceptual separation can involve temporal, spatial or technological separation.

The simplest counterexample is that of Turing's Machine and other pre-compiler computing systems. These computers were developed to be machines for working with complex numerical data, and the processes for controlling them are heavily couched in terms of 'operating the machine' (Rettig, 2003, p. 5), (e.g. Dijkstra, c1997). Computers were operated using a variety of programming techniques, which do not appear to have been seen as 'languages' at the time, though in hindsight we can easily treat them as such.

The first programming technique (for general-purpose electronic computers, at least) was the system of plugging patch cables into sockets on computers like the ENIAC (Electronic Numerical Integrator And Computer) in 1945. If it were treated as a programming language, with attendant lexicon, syntax and semantics, the language lexicon would consist of dozens of sockets and cables, and around three thousand switches (Sanford, 1997); the syntax would be imposed by the electrical compatibility of the connections, and the semantics would be expressed in terms of Boolean operators and predicate logic written on paper. The semantics of the language directly map onto the electrical functions of the logic gates and accumulators, implemented on the 18,000 vacuum tubes in the machine.



Operators programming the ENIAC by patching cables.

Whilst such a language would perfectly 'comprehensible' to machinery, it was by no means as straightforward for the programmers, who had to think long and hard on paper about even basic algorithms and optimisation, and, once this was completed, take the considerable time (up to two days) to patch everything together (Sankaran, 1995). Programmers were pioneers and researchers.

It was not long before a more pragmatic way of programming was devised, in particular with the introduction of stored-program computers, as a result of the 1945 work of John von Neumann (von Neumann, 1945). The programs were stored in magnetic wire and, later, punched on a card, which meant that the program could be quickly reloaded when necessary. EDVAC (Electronic Discrete Variable Automatic Computer), the first stored-program computer development, became operational in August 1949 (Andersen, 2002). To summarise, in these early years, the only use of a computer was to program it with switches and cables, and later with punched cards—there was no difference between programming a computer and using one, because every use of a computer was programmatic.

An important development around 1958–62 (after Hopper's compiler) was John McCarthy's LISP. In McCarthy's own words:

> *LISP is characterised by the following ideas: … representation of symbolic expressions and other information by list structure in the memory of a computer, … a small set of selector and constructor operations expressed as functions, composition of functions as a tool for forming more complex functions, … the recursive use of conditional expressions as a sufficient tool for building computable functions, … the representation of LISP programs as LISP data, … the LISP function eval that serves both as a formal definition of the language and as an interpreter, and garbage collection as a means of handling the erasure problem. … Some of these ideas were taken from other languages, but most were new.*
>
> (McCarthy, 1981)

LISP has a very small and elegant language definition, because LISP code is written in the same format as the lists it manipulates—the S-expression syntax with its characteristic parentheses. In classic LISP, the code to evaluate a number's factorial looks like this:

```
(defun factorial (n)(cond ((zerop n) 1) ; if n=0, return 1(t
(times n (factorial (sub1 n)))))) ; elsereturn
n*factorial(n-1)
```

LISP is the first language to implement the now-ubiquitous if–then–else command. Like the Turing Machine, it is the implementation of what started as a thought experiment, but due to its elegant definition, and its close fit with von Neumann architecture (instructions and data being stored and manipulated similarly), LISP is a more convenient alternative to the Turing Machine for research in computer science, with particular strength in the emerging field of artificial intelligence.

As Paul Graham points out (Graham, 2002), Lisp and Fortran (both written in lowercase nowadays) are the only two 1950s languages that are still in use today. Graham argues that Lisp's longevity lies in its mathematical simplicity (because mathematics doesn't date like technology), whereas Fortran's lies in its closeness to assembly language (which doesn't change as rapidly as higher-level, feature-specific languages). Alan Kay (Kay, 1993) wrote,

> *all [programming languages] seem to be either an "agglutination of features" or a "crystallisation of style". COBOL, PL/1, Ada, etc. belong to the first kind, LISP, APL—and Smalltalk—are the second kind. It is probably not an accident that the agglutinative languages all seem to have been instigated by committees, and the crystallisation languages by a single person*
>
> (pp. 69-70)

Kay's 'crystalline', elegant languages, reflect the simplicity and flexibility of the computer in a way that the 'glutinous' languages are incapable of, because the latter reflect the constraints of multiple stakeholders, even before anyone does any creative programming with them. Alan Kay is one of the developers (in the 1970s, with Dan Ingalls, Ted Kaehler and Adele Goldberg (Ingalls, Kaehler, Maloney, Wallace, & Kay, 1997)) of the Smalltalk language to which he refers. Smalltalk has many 'flavours' today, but one in particular is the clearest counterexample to the compiler wall: Squeak Smalltalk. In order to understand why Squeak Smalltalk is so important, we must first look at Smalltalk itself. Smalltalk is designed to be a language for kids to use to learn programming. Alan Kay, in a conversation with Karen Frenkel (Frenkel, 1994) summarises his argument for kids learning programming with a metaphor:

> *Children in the US are almost certainly going to grow up to drive cars. So why don't we, at the age of two, put them in little motorised vehicles so that from the age of two they can learn how to be much better drivers than if they just start at age 16? … From my standpoint this is horrible because the kids' muscles would atrophy. On the other hand we can give kids bikes, which wouldn't bother me at all because a bike is something that allows kids to exercise … So the questions we have to ask for any kind of software is when is it a bike and when is it a car? And when should it be a bike and when should it be a car? Giving kids software that is like the car software that adults want to use, which is basically prepackaged solutions to things, is disastrous. What kids need are challenges*
>
> (Frenkel, 1994, p. 16)

Kay's vision was then embodied in the Dynabook, nearly a prototype of today's laptops, and today his and Papert's (Papert, 1980; Papert & Harel, 1993) ideas about constructionist learning inspire the One Laptop Per Child project, which aims to provide networked laptops costing roughly US$100 to every child on the planet (http://laptop.org/).

Smalltalk was an early object-oriented language and Kay is regarded as one of the fathers of object-oriented programming. Kay wrote this about his conception of the term "object-oriented":

> *… each Smalltalk object is a recursion of the entire possibilities of the computer. Thus its semantics are a bit like having thousands and thousands of computers all hooked together by a very fast network … For the first time I thought of the whole as the entire computer and wondered why anyone would want to divide it up into weaker things called data structures and procedures. Why not divide it up into little computers, as time-sharing was starting to?*
>
> (Kay, 1993, pp. 70–71)

Here he invokes a Lisp-inspired recursive design, which is aimed at simplifying understanding of the structure of computing by decomposing that structure into several self-similar parts. Like the Turing Machine and Lisp before it, one only has to have enough skills to understand one part in order to have enough skills to understand the whole computing system, if one explores far enough. This recursive design removes much of the conceptual separation between programming and using, because deep programming works in the same way as gentle programming (as we now characterise 'using').

Object-orientation is an elegant way to program that encompasses the creative possibilities of a computer, but if it is compiled conventionally (as it is in C++, C#, Objective C and Java) then the disadvantages of the compiler wall remain. Smalltalk, however, is also an environment—originally envisaged as an operating system, but now existing as a cross-platform virtual machine, which allows manipulation of Smalltalk objects in real time, in run-time, with simple graphical interface manipulation (so no command line). Since everything in a Smalltalk program is an object, and every object can be accessed and modified at

run-time, the entire program can be modified without having to be recompiled. Had Photoshop been made in Smalltalk, it would have been a relatively simple matter (i.e. possible) to inspect and modify the Blur tool object, or its sub- and super-objects, even while the Photoshop application was running. Smalltalk is the counterexample we are looking for which shows that programming computers need not be separated from using them.

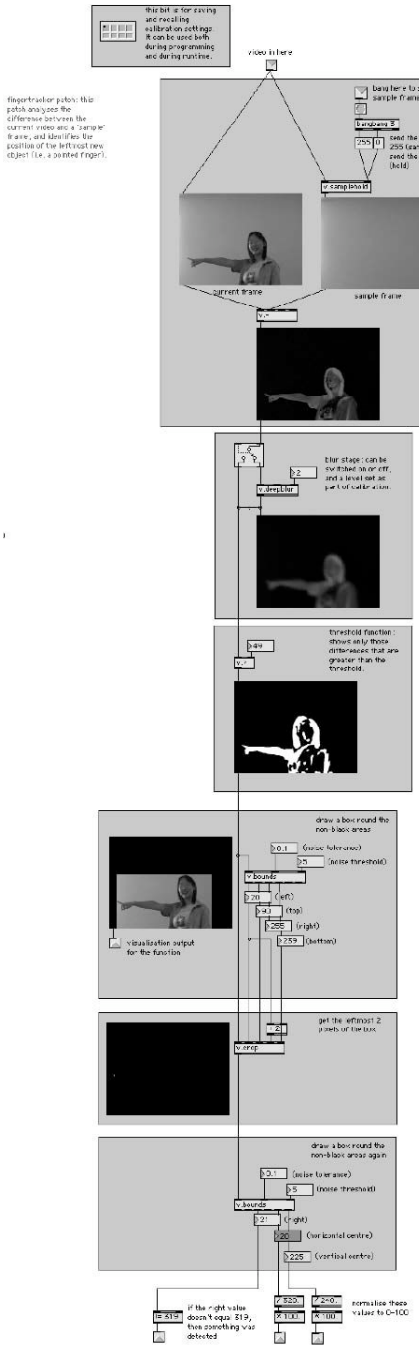
Squeak Smalltalk goes an important step further than reducing the conceptual separation between programming and using enforced by the compiler wall: I said before that Smalltalk's object-orientation means that one only has to have enough skills to understand one part in order to have the skills to understand the whole system. However, having the skills to understand the whole system is not the same as *actually being able* to understand the whole system. One still needs to be able to access/explore the whole system, and this is where Squeak Smalltalk comes in. Squeak Smalltalk is written in Squeak Smalltalk, including the virtual machine that runs Squeak Smalltalk programs. This means that every aspect of the (virtual) machine, not just applications, is a Smalltalk object, and can be accessed and modified as it runs. This is the epitome of creativity support technology as I have described it—a computing medium with constraints flexible enough that any possible change can be made to its sub-media in support of creative conceptual spaces. For instance, it is even possible, should it be desired, to redefine the meaning of the mathematical operations Cosine and Sine for the entire Squeak Smalltalk environment.

Yet Smalltalk is ultimately is a textual language, which describes computation in a spatially-detached manner. The spatial separation that compilers impose between programming and using can be overcome to a greater degree than with Smalltalk. Max/MSP is a visual programming environment, popular with interactive artists, and is based on the Max programming paradigm, with the addition of MSP objects for synthesis, created by Miller S. Puckette (M.S.P.), who describes it thus:

> *The Max paradigm can be described as a way of combining pre-designed building blocks into configurations useful for real-time computer music performance. This includes a protocol for scheduling*

control- and audio-rate computations, an approach to modularisation and component intercommunication, and a graphical representation and editor for patches.

(Puckette, 2002)



Max's strength lies in its separation of *real-time* control systems into self-contained, parallel processes (represented by box-shaped 'objects'[6] on screen), rather than forcing the programmer to decide in what order events should be dealt with (but allowing him to do so should he wish). Some objects perform calculations, some objects gather input from the mouse, keyboard or other sensors, and some objects control graphics, audio and other displays. Max works by passing messages to and from such objects through virtual 'patch cables' (recalling analogue synthesisers and early computer programming). Thus, Max programs are called 'patches'.

Max patches have no distinction between edit-time and run-time[7] . The separation of Max's objects means that, as in Kay's description of Smalltalk's objects, each one acts like a mini-computer, which carries out a self-contained function on an as-needed basis. For example, Max's '+' object adds the value at the right inlet to the value at the left inlet, and sends the result out of its outlet, whenever, and only whenever a value is received at the left inlet. As soon as a '+' object is created, it *always* has this behaviour, no matter the extent that the user-programmer is changing or using the patch.

By programming with a concrete graphical representation, manipulated chiefly by mouse drags, Max reduces the conceptual separation between programming and using. By combining processing objects and input/output objects in a patch, Max exposes the program at the same time as the user interface, and hence removes the spatial distinction between programming and using. By treating objects as always-running mini-computers, Max, like Smalltalk, removes the temporal distinction between programming and using.

So Smalltalk and Max/MSP between them (see Appendix 2 (on CD) for heuristic critiques of both) show that the separation between programming computers and using them is not fundamental to computing, but rather is a consequence of conventional compilation technology.

---

6 not the same as Smalltalk's objects, though they could conceivably be considered a subset.

7 A patch can be 'locked', which protects patches from accidental changes and which allows parameter controls to be manipulated more easily, but this does not permanently prevent change in the way that compilers do

There are many other such counter-examples—spreadsheets being the canonical end-user programming example (e.g. Nardi, 1993), but these illustrate the point nicely, and together show us a way forward (see Chapter 8 for a potential synthesis).

Hardware

All of this writing about programming and software technology has paid scant attention to hardware, which, whilst physical and scrutable in a way that current software isn't, presents the true fixed boundaries in which all software, and thus all creative software, must operate. On one hand, hardware is the rigid constraint of computing, which creative workers may choose to use or not use in favour of another form (such as electronic engineering). On the other hand, Alan Kay calls hardware "just software crystallised early" (Kay, 1993, p. 87) and laments its presentation to programmers "as a given"—with no attention paid to making it appear reasonable to end-users. Someone who transitions to sufficiently deep programming will discover these flawed aspects of hardware, and so may then be motivated to transition 'deeper' into hardware design. Such developments would be outside the scope of this thesis, however.

Open Source Software

During parts of this discussion, some readers may have wondered about open source software's role in creativity support. Open source software (the genre, not the political movement) is software for which the source code is explicitly made available. Generally this means that, in contrast with closed source software, if a creative user wants to change a part of some open source software, and they are able to understand its source code sufficiently, they are free to change the software in accordance with their wishes. The originators of the software frequently require that the changes are made similarly available.

In conjunction with the Internet, open source software has allowed developers to produce complex and well-regarded software (such as the GNU/Linux operating system or the Apache web server), by changing open source projects and sharing the results. Amongst some hackers, having others able to see their code leads them to take pride in high

quality, quick work. This, combined with a lack of a single chain of command, can mean that innovations or other changes can appear in open source software more quickly than they might in closed source software (von Hippel, 2002).

And no clearer example exists of open source spawning rapid innovation than in the World Wide Web. Web browsers receive the information they need to display a web page as text files (predominantly HTML, XHTML, Javascript and CSS) and image files (mostly PNG, GIF and JPEG). This convention means that it is straightforward for web designers to dissect any given web page in order to figure out how a particular feature was implemented. Any page or part of a page can be copied and then modified by any web designer for use in his or her own projects. This has resulted in new ideas being quickly proliferated, documented and modified, regardless of the wishes of the idea's originator.

But open source is not the panacea that some see it to be. Whilst the open-ness of the source code is supportive of creativity, because it allows computing media to be unconstrained by its designers, modifying that source generally requires ability to deep program. Unlike Max/MSP, an environment where program modifications can be made in a similar mode to gentle interface manipulations, most open source software is written in a conventional textual, compiled programming language, with the attendant conceptual, spatial and temporal separations, as I have described before, between programming and using. Squeak Smalltalk is an open-source system that removes temporal and some conceptual separations, but, being textual at heart, retains the spatial separation.

So why do we compile software?

Originally, there were extremely few programming languages; now there are thousands, if not uncountably many, depending on one's view of what it means to program. The perception of a separation between programming and using began to emerge when Grace Hopper began developing the first automatic compiler in 1951, called A-0 (which later developed into Flow-Matic and in turn COBOL). This compiler translated mathematical symbols into machine code automatically, whereas prior to that, machine code had been created by hand. Flow-Matic, along with ALGOL 58 and the popular FORTRAN I, formed the van-

guard of the new programming languages. Each of these was tightly tied to mathematical expressions and descriptions of algorithms. This was not only because of the types of problems that computers were then employed to solve, but also due to the high cost and low speed of computers meaning that efficiency was a big concern.

So at that point, there were languages that were closely tied to hardware operations. As we follow the path of programming language development into the second generation, we see that, there is still no 'use' of a computer that didn't involve programming in the strict sense. However, different programming languages were arising for different applications. The language COBOL, developed by Hopper after Flow-Matic, was the first to be developed specifically for an application other than mathematics, and so used the language and conceptual space of its target domain, business. Hopper herself believed that:"… the major obstacle to computers in non-scientific and business applications was the dearth of programmers for these far from user-friendly new machines. The key to opening up new worlds to computing, she knew, was the development and *refinement* of programming languages—languages that could be understood and used by people who were neither mathematicians nor computer experts" (McKenzie, 1994, my emphasis).

Though Hopper (via McKenzie (McKenzie, 1994)) talks about opening new worlds of computing, she rather presages the 'modern Babel' of refined and esoteric programming languages and applications; closed worlds rather than open. Implicit in this is the development of millions of applications from this multitude of programming languages. Since applications are themselves 'gentle' programming languages, there are millions of languages in which to do computation, and very few of them can communicate with each other, or change in any way that we can envisage.

We have dealt with the historical origins and rationale for compilers. There are also pragmatic reasons to keep compilers as one-way translators of code, the way they have been since Hopper's time. Two obvious reasons are optimisation of execution speed and efficiency of size of program. I will discount the latter as a relevant reason on the assump-

tion that storage space is plentiful enough for the creativity support benefits of two-way compilation to be worth the extra space. As for speed of execution, it should be possible to compile an optimised program in the background. We should consider reflecting some of the other underlying concerns in the designs of two-way compilers.

For example, one obvious reason is that, by compiling the source code, it can be obscured from people who the program designers wish to keep from seeing that code. This is normally in order to protect the Intellectual Property (IP) embedded in the code from being copied, or to prevent potentially unscrupulous people from examining the code to find security flaws. IP is potentially lucrative for software companies; the risk of harm caused by security flaws in software is potentially damaging. However, this means that people interested for interest's sake (which, as we discovered in the literature review (see pp. 32-36), creative people are likely to be) are denied such knowledge, even if they have no intention of commercially exploiting such knowledge. In this respect, software is markedly different from physical objects, which cannot be protected from dismantling, exploration and customisation.

Another advantage of hiding source code stems from the ability to support the compiled program's users. If every user of a piece of software runs the same or similar versions of it, the task of locating a problem via email or the telephone, is made easier. On the other hand, if every user is able to rewrite portions of the code, it becomes difficult to tell whether a given problem is caused by the software provided to the user, or by the user's modifications.

Yet the increasingly embattled efforts of IP owners to prevent copying for any purpose, combined with the powerful laws in place to discourage illegal copying, also combined with the increasing success and strengths of open source software mean that technological protection, by compilers, is becoming decreasingly relevant. As for the problem of supporting home-modified portions of code, it would be a simple matter for a software provider to adopt a policy to support unmodified code only.

Summary

As our conceptual spaces change through creative activity, so too do our requirements of the tools we need to support our conceptual spaces and creativity. If our tools are software, then changing software is the best way to keep tools in pace with our changing conceptual spaces. Yet we are usually prevented from changing software—not only because programming is hard, but because software is compiled, closed, rigid, static, when it could be de-compilable, open, flexible, dynamic. Today we are given a world where programming and using, compiling and runtime are entirely separate, exclusive situations, as characterised by technology, and the way we are supposed think about technology, as though it was fundamental, or innate, or natural, or necessary. But if that separation is truly fundamental to computing, there would exist no systems that didn't enforce it. Therefore, if any system can be found which does not enforce such a separation, then that separation would not be fundamental, but rather a particular mythology. I have shown examples of these systems.

While the separation reflects particular pragmatic concerns in software design, these concerns need not dominate as they do now. The separation of programming and using is at the heart of a false dilemma in contemporary computing, and it is critical to overcome that false dilemma in order for computing to truly support creativity in accordance with its potential.

6. Study 2, Part A—Technologists' Roles in Interactive Art Collaborations: Analysis of Previous Data

Introduction

This is the first part of my second study, a grounded theory study of the roles of technologists in interactive art collaborations. The methodology of the study is explained in Chapter 3. The specific aims of the study are not to be specified at the outset, as the point of grounded theory is to attempt to 'let' the theory emerge as the study progresses, whilst minimising the researcher's preconceived notions. However, I am generally researching creativity support, and I have established the hypothetical creativity support potential of computing in the previous chapter. What interests me most, then, are the ways that interactive art is developed in these interdisciplinary collaborations, and in particular, the ways that a computer's creativity support potential can be conveyed to an artist. If an artist doesn't program, yet collaborates with a technologist who does, then presumably the technologist has a significant role to play in conveying that potential.

The study is split into two because it has a long description and, unfortunately for the reader, the first part of such a description is rather tedious. Grounded theory involves many iterations of data gathering, coding and analysis, beginning with broad open coding and refining to selective coding and detailed memos. As such, the beginning of the analysis will be fragmented and rudimentary, and so this chapter may be skipped if the reader is not interested in these early, diffuse, stages of analysis. It is not until all of the codes are drawn together and written up that a coherent theory emerges.

As Chapter 3 describes in detail, the first iteration of coding was on the case reports from the COSTART study of art–technology collaborations, from which open coding took place. The second iteration was on interview transcripts from the COSTART 2 study, which, together with the coding from the first stage, formed some categories. These studies concentrated on qualities of collaboration, rather than technological issues per se. Subsequent iterations of analysis were based on new interviews I conducted with artists and technologists in 2004–5. This chapter describes the COSTART and COSTART 2 coding and analysis, and the next chapter describes the coherent theory resulting from the analysis of the interviews.

The coding and categorisation of the COSTART data, with editing for confidentiality and relevance (sections with no codes are omitted) can be seen in Appendices 3 and 4. The rest of the chapter is essentially a commentary on these appendices.

Codes, and categories formed by many codes, are underlined in what follows. 'A' is short for artist, 'T' is short for technologist and 'O' is short for the third-party Observer in the COSTART studies.

COSTART Reports

The first passage that was coded in the COSTART reports was this one, from report 2 (Candy & Kelly, 2000b):

> *A2: [T2] suggested that different effects might be achieved that would be equally interesting … I felt my creativity was lessened if different systems were used and this was a problem … because I*

> *know that [my idea] would achieve the effect I required whereas other systems might not …*
>
> (p.4)

Immediately, we can see a process of suggesting/rejecting changes to the artwork. This suggestion/rejection is a form of communication (category) between artist and technologist (and as we shall see, permeate throughout the analysis).

The other codes are that A2 requires 'effects' and that A2 wants to 'retain' creativity. Requirements are probably important in artworks, and communicating these requirements could be important in collaborations. Retention (or ownership?) of the creativity is an interesting concept to monitor.

The next coded sections deal with T2's revisions of time estimates and difficulty estimates. These could be issues of project management in creativity support, or could result from poor communication of requirements. The collaboration is described by T2 in terms of 'problems'. Is this a detached problem/solution use of the word, or a more negative use of the word? (I think it is negative, because of the difficulties described in this collaboration).

The next passage mentions communication explicitly:

> *T2: Difficulty communicating. Sometimes I believe I've resolved the problem with [A2] but the next time we talk it seems he meant something else. It seems difficult to impress upon [A2] the need to look at the problem logically and be very specific. This is not his fault but the problem must be expressed in sequential logical terms … It was very difficult to express [a preferred method] in non-threatening terms i.e. where [A2] did not feel he was being forced to compromise for the convenience of the support team [i.e. me]*
>
> (p.8)

This is the first description of a disconnection in styles of thinking. The technologist 'needs' the artist to express the 'problem' in a logical, specific manner, whereas, according to the technologist, the artist wasn't doing so. The conflict is so strong that the technologist feels that to pro-

pose a sequential approach could easily be threatening, or forced. At the same time, the technologist experiences internal conflict—a feeling of guilt at forcing a technological compromise. Moreover, the technologist clearly feels he is in a subordinate role (as the "support team"), yet also feels he must direct the artist's thinking process.

This passage, whilst not to be extrapolated into all art–technology collaborations, raises several flags. First, we must keep an eye out for the variation in approaches that an artist and technologist might have to a given work, and the ease with which those approaches can be communicated. Secondly, we should be aware of the conflict experienced internally by both parties, as well as conflicts in external communication. (I decided here to keep a track of which codes originated from artists, and which from technologists, since it became apparent that they would not always align).

The next passage continues this theme, but (the technologist) reframes it in terms of rules. Here there appears to be a shared understanding that 'rules' are important, but, by itself, the artist is unable to specify the rules. Here, the key quote is the technologist's comment about the artist:

> *it can be quite difficult to get those rules because you know quite clearly what you want and it can appear that I know what you want as well, but when it comes to specifying it to the computer, it is different*.
>
> (p.13)

The communication in this collaboration is three-way. Even though the artist and technologist might appear to understand each other, it only becomes apparent that they don't when the computer mediates the communication. What counts, in this collaboration, is whether the shared meaning is expressible in computational terms, and we need to see whether this is the case in other collaborations.

Finally, in this COSTART report, we see the artist's views on programming.

> *A2: To insist that the artist learn how to program software would constrain a lot of initial ideas and possibly lead to the abandoning of proposals. A step by step approach to tackling issues was very useful and a bank of solutions could be presented as required for new artists. New ideas, however, might mean that there is no bank of precedents for art works previously created.*
>
> (p.23)

In relation to their previous discussions, the artist describes the 'step-by-step' (sequential, logical) approach as being useful—but the precise nature of that usefulness is unclear, given the previous conflict in styles. In fact, in the same passage, the artist feels that programming would constrain ideas (in other words, the artist is not prepared to have his or her conceptual space constrained by the limitations of programming). The artist appears to think that the step-by-step approach would be useful for little more than presenting a bank of solutions, not for exploring new ideas. This may be a side-swipe at the technologist's approach in the collaboration, or may be a view more held by the artist in a more general sense.

Moving on to the next COSTART case, number 3 (Candy & Kelly, 2000c), we see another difference of approach.

> *A3: I think what [T2] is doing is very complicated and I don't think I'd be able to mess around with it. What I was thinking was to work around it, not really do anything in it …*
>
> *O3: [T2] was working on the idea of giving you something to take away with you that was kind of complete.*
>
> *A3: Yeah (laughs)*
>
> (p. 6)

The context here is that the artist wanted a toolkit for making art with (by "messing around"); the technologist wanted to construct the finished product, the art itself, and was 'hard-coding' the art. This points at the artist's reluctance to delegate control over an artwork, or conversely,

the artist's need to spend a lot of time working directly with the developing artwork.

The next passage is the technologist's description of artists (in general) misconceiving that computing resources are unlimited and 'bear no cost'. This can be seen as an allusion to understanding *servitude*, one of the four Ss of computing described in the first study (Chapter 5). The technologist feels that artists could benefit from an understanding of the limits of a computer's servitude. The passage is also interesting because, in contrast to the previous report, where the technologist's stated goal was to 'solve problems', here the stated goal is to get 'the image [the artist] want[s]', which could be a somewhat less reductionist way of approaching the work.

The next passage clarifies some suppositions of the first passage.

> *O3: … you sent [said?] something to the effect that it wasn't very satisfying to do something through the eyes and hands of someone else…*
>
> *A3: I felt I was being misled and I was pretty sure that I could get what I wanted but [T2] was not letting me have it (laughs)…*
>
> *O3: So, you'd rather have been doing it yourself to be confident about that?*
>
> *A3: Yes, because unless you see it not working then you don't accept that it can't work.*
>
> *O3: So that relationship of having someone take on certain parts of the task really took away from you the ability to check.*
>
> *A3: Yes because when you do work with editors in film for example and you're the director, the relationship is different. They may say 'well I don't think it's going to work' and you say 'well, can you just try it'. And they try it and maybe it works and maybe it doesn't,*

> *but it often does work because your hunch is right. And so everyone moves on…*
>
> (pp.18–19)

The artist looks for satisfaction, and doesn't find it by creating the work 'through the eyes and hands of someone else' (p. 18). This dissatisfaction is strong enough for the artist to feel 'misled' when the technologist was not letting the artist have what she wanted. The reason for this dissatisfaction could be that the technologist knew that such-and-such an approach wouldn't work for sound technological reasons, so dissuaded (or prevented, by inaction) the artist from exploring that approach. Exploration (and thus experimentation), according to the literature (see Chapter 2), is an extremely important part of creativity, and is used to discover constraints. By dissuading the artist from exploration, the technologist imposes his own constraints—the artist is prevented from discovering the nature of the underlying technology that limits the exploratory approach.

The coding of the next case, number 4 (Candy & Kelly, 2000d), starts with one technologist paying tribute to another's domain knowledge. This passage shows that not only is domain knowledge important (as we know from the literature), but also that technologists may rely on other technologists.

The next interesting passage is an excerpt from an interview with the artist about her relationship with technology (p. 12). On her home computer, the artist is limited by the capabilities of the technology (specifically the hard disk space available), but it does not appear to constrain her ability to execute ideas, even though they are "working ideas, and they are not going to go further". In a sense, the artist has acknowledged a constraint of her chosen sub-medium, but has not let that constraint constrain her conceptual space; she chooses rather to downgrade the quality. Since the compromise was technological (and, in this case, transcendable), the artist sees the compromise as being 'wrong', because of the conflict with maintaining quality. We can also relate this passage to the earlier passage about understanding the limitations of *servitude*. In the case of hard disk space, the operating system has a built-in gauge of hard disk space, and the artist could have used this to figure out what

the nature of the limitation; in the earlier case, the problem was processing related, specifically the polygon count of rendered objects lowering the frame rate. Since there is no hard limit on either polygon count or frame rate, in the way that there is on hard disk space, the trade-off maybe harder to understand. This could have been compounded by lack of explicit monitoring of either by the computer, or the reluctance of the technologist to implement a high-polygon, low frame rate, scenario.

COSTART Report 5 (Candy & Kelly, 2000e) corroborates the earlier finding about different approaches. Here is the relevant passage in full:

> *T5: What's the effect you are trying to achieve. What's the mood you're trying to achieve?A5: So that it's hanging in space. It's slightly flat; it's really trying to force that interpretation.*
>
> *T5: What sort of lights are you looking at …what's your light basement in your concept, where all the lights are placed and what sort of lights also.*
>
> *A5: I don't know, I mean really it's manufactured, you can't really say what sort of lights it doesn't matter…*
>
> *T5 (interrupts): Spotlight or…*
>
> *A5: But what I'm saying is it doesn't really matter what sort of lights they are. What I'm talking about is the effect, do you know what I mean? I know on this, yes, there's a spotlight, but in actuality they're not and I'm not thinking of it as being, how actual lights would be because it's just to make it manufactured to sit there. It's really experimenting to try and bring it out.*
>
> *T5: OK…That sort of thing?*
>
> *A5: I think actually it's more of, so that it would actually be cast… You see there at the moment the second object is sitting like that. If*

that could be like there…So not that it's getting wider but it's getting narrower. More of a spotlight…because there's already one big one isn't there. What's the other one doing?…

T5: That's catching this wall.

A5: But that's not getting any light…

T5: Yeah, but you want that reflection that is a reflection of light.

A5: Yes, but it's not really relevant to that because there's nothing that is reflective.

T5: But we can make it reflect… for example this thing we're looking at, this thing, it's basically catching the stream light (B agrees), and then it's reflecting that light you see.

A5: But what's the difference between that and what you're doing there?T5: It's not reflecting light it's reflecting the object…So do you want it to show the light?

A5: I just want to…use light to show the form…I just want to communicate what's within it.

T5: What I could do is add the light object like this stream light, and take that shape and reflect it as a reflection of light.

A5: But that's the same from what you were trying to do before.

T5: It's slightly different. It's different because it's not catching the light; it's catching the object.

A5: But it would just be light though, wouldn't it? It's supposed to be a void. Can you make it so that it's white rather than yellow?… Just that last one, you know the little one…Will it not do it so that it actually comes out as a patch of white?

T5: No, but it like to assume realistic conditions, so you've got a green and a blue light and if you project a white light it won't be white it will mix with the green and the other lights overlapping there.

A5: So how do you get white?T5: Well, you can't get that in the real world…we'll have to paint the texture, on the texture itself… Or, I make the texture more white in it's nature

A5: …what it's doing is it's transferring it to an acidic colour which flattens, do you know what I mean…whereas if there were sort of white patches. Like sort of small subtle white patches then that would spring out from the yellow.

T5: Yeah, but…It is a white light and it is bouncing off the yellow object, so it's yellowish. You tend to feel it is yellow, it is not yellow.

A5: But that's how it looks…you know it looks as if it's burning, it's bright yellow…

T5 interjects: Look—yeah, that is because of the texture. The texture is yellow. The light is white…

A5 tries to explain again the effect that she is looking for.

(p.7)

The passage starts promisingly enough, with the technologist asking about the desired "effect" and the "mood" (as opposed to the 'problem',

the 'rules' or the 'image' seen in earlier reports) that the artist is trying to achieve. The artist responds with a similarly poetic description ("hanging in space … slightly flat … trying to force that interpretation"), but then the communication goes wrong. The technologist, possibly realising that he couldn't interpret such an answer in computational (virtual reality) terms, asks instead "What sort of lights … spotlight, or …?", even interrupting the artist when the answers are not of the type that the technologist is looking for.

I cautiously begin to categorise here. In all the cases we have seen of different approaches to describing the artwork, the technologist has been asking questions of the artist, and feels compelled to ask the questions in computationally-meaningful terms (we have seen requests for 'logical', 'sequential', 'rules', and 'realistic' lighting specifications), whereas the artist describes (and, we must suppose, thinks about) the art in more subjective terms. At some point, a translation needs to take place from subjective terms to computational terms—in successful collaborations, how, and where, does this translation take place?

The next passage is from an interview with the artist:

> *A5: We (T5 and A5) found a balance of the best way to go forward. I've learnt loads from T5. I'm used to sitting in front of a computer all day and knowing what I'm doing with the technology that I'm using all the time. So to me it's quite strange coming in and sitting and watching and not being hands on. Because I'm very much if I'm learning something I'll be asking someone else and doing it. I found that quite frustrating at first of all also when you're trying to make an object a particular shape and things like that it's easier for me if I can do that myself … because I don't like saying to someone 'in a bit on the left' and that sort of instruction because I feel like I'm being really pedantic. I feel like I'm a bit, you know, when there are things that are really important to you.*
>
> *O5: But there were some things that you could have been doing yourself.*

> *A5: Yeah and that's what we did. I think we both thought this is going to be better if we have us (referring to herself) doing things like that. How it's ended up is I've been doing a lot of the hands-on work and T5 has been running through a way that's going to be appropriate to develop it and then I come back to him. Goes stage by stage. It's worked out really well. I've learnt loads." (p.18)This passage reinforces the passage in report 3 about the artist feeling uncomfortable, passive and frustrated by not operating the computer directly, but rather directing someone else. Here the artist doesn't like feeling pedantic about directing the minutiae of the development.*

To diffuse some of the artist's frustration (and possibly that of the technologist being 'micro-managed'), the team has worked out ways for the artist to do some hands-on work, and for the technologist to develop the work in a stage-by-stage process, where each can work separately. It will be interesting to track the relationship between one-on-one collaboration and working separately.

Finally, there is a brief comment on learning from the collaboration. This is recapitulated in the next passage:

> *A5: It's been a huge help. It's something I've wanted to do for a long time on the learning side of things, to learn some 3D whatever software to get a grip of the actual process. You can get so far but you can't get as far as if you're sitting with somebody else.*
>
> (p.19)

So the process of 'sitting with somebody else' (one-on-one) collaboration appears to be a useful learning process, both of software skills and, possibly, of collaboration processes themselves.

The next report, number 6 (Candy & Kelly, 2000f), begins with a statement about important qualities of collaborations, and describes a particularly supportive technologist:

> *A6: I can only speak from a personal point of view and say that I have never before experienced the quality of support that I received*

> *from COSTART. I would rank people skills as paramount…The intelligence, goodwill and sensitivity of the support team is vital. I consider myself very fortunate to have had the best and most appropriate kind of support for what I wanted to achieve.*
>
> (p. 7)

We can find some basis for those qualities in the text that follows. For example, the observer notes the process of requirements gathering in the collaboration:

> *O6: A6 wants to know if it is possible to measure the complexity of the figures he works with in his art. T6 explains the notion of complexity and how it is measured. This leads to a discussion in which T6 points out that one of the virtues of the computer is that it forces explicit decisions and helps to reveal options not previously thought of by just looking or thinking. T6 points out that a program can be written to do transformations from A6's base figure to other levels. (A6 seems to not have realised before that this is possible.) In looking at one set of A6's figures T6 begins to ask what counts as critical detail, e.g., no rotations, circular sequence, no repeats allowed, no inverses, none are uniformly the same value. (T6 seems to be extracting from A6 the knowledge he needs to begin writing the program.[…]*
>
> *T6: Well I think I have a number of things to address. One is to start to map out the decisions that are made…There's an advantage of using the computer sometimes which is that it makes you be explicit about decisions that are made and sometimes what you do in the computer in the end doesn't' matter very much because…One could generalise this kind of work to say well to come down to this particular moment I've made this that and the other decision. Doing that would reveal that there were other options that you haven't actually thought of become obvious and that could lead to new work, which may or may not be executed using a computer.*
>
> (p.14)

Here, the observer notes that the technologist is describing virtues of computers as being that they force explicit decisions, and can reveal options not previously thought of. In this collaboration the technologist is using the computer to show the artist new possibilities ('other levels'), rather than to implement the artist's pre-existing idea. Perhaps the technologist's exposition of the constraints of the computer is a factor that made the artist feel more supported. The technologist is also actively asking about what counts as critical detail, and extracting knowledge from the artist, as opposed to simply passively responding to instructions. This could well be a process of creative engagement by the technologist; the artist is likely to be enthused by the such engagement.

Later in the report, the process of exploring new possibilities is documented.

A6 asks about the possibility of incorporating sound

> *T6: Well what I suggested earlier in the week we've started to build a core here and this is like the little engine in the middle and now we can add all things and use it."*
>
> (p.16)

The technologist is describing the program core as an 'engine' to which things can be added. This is a sign of good creative computing—that the technologist has built a system with unforeseen expansion in mind. How useful or novel the engine analogy is to the artist is unclear.

Finally in this report, the discussion touches briefly upon the learning process:

> *T6: That's what the code does…this is just the core bit of the program—this is the important bit. Just to take you through this quickly. I don't know how much you obviously know about it… Well it's a question of whether you want to end up writing this kind of stuff or not. A6 confirms that it would be beneficial to learn something about the programming. T6: Shall we look at the program working first and then we can go through this later. Would that be OK?*

A6: Yes

14:53: T6 starts to show A6 how the program works by taking him through step by step…"

(p.18)

The artist expressed a desire to learn 'something' about programming (this is the observer's interpretation, so what the 'something' or 'benefit' is ambiguous). The step by step description of the program code, and the artist's response to it, is also undocumented.

Case Report 7 (Candy & Kelly, 2000g) carries little information about the technologist's role in the collaboration. The two important passages are:

A7: Also talk about sensors. Trying to adjust thinking to work more closely with programming. Using digital tech to achieve an end in itself—or using dig. tech. as a creative element in itself—confusion over this—want to make unexpected or unplanned animations but still control overall style.

(p. 7)

and, O7 writes:

A7 describes effect. T7 poses questions to clarify how it might work under different conditions. (The questions do get A7 to think about choices she hasn't made yet so then she begins to explore these choices.) (Helps to define start states and operating conditions) p.8: T7 and A7 are looking at some drawings, thinking about the sequence of the piece—how it will begin and continue to evolve in accordance with the changes in weather and environmental conditions…A7 explains that she is finding it difficult and confusing to cope with the amount of decisions to be made because there are so many possibilities. Just in terms of imagery. She says: "It's like a set of characters and in some way they are going to blend together depending on different circumstances." Each image has an identity

> *within itself and each has to relate to the other in a whole host of different combinations.*
>
> (p. 7–8)

The first passage, from the artist's diary, describes the artist's confusion over whether to use digital technology as a means to an end, or as a creative element in itself. The artist is also attempting to adjust her thinking to 'work more closely with programming', though she does not seem confident of her ability to do this. In the last section of the passage, the artist appears to want to define the limits in which the computer can be unpredictable. In the second passage, the technologist is attempting to clarify the intended behaviour of the piece, by asking questions about the behaviour under various supposed conditions. This question-asking is similar to the requirement-gathering processes in other reports. The observer remarks that some of these conditions were unanticipated, and this prompts the artist to explore her choices (more carefully).

The artist is finding it difficult to make decisions, because (according to the observer's report) there are so many of them, representing many possibilities. There seems to be a lack of ways to explore the possibility space.

That brings us to the end of the COSTART I reports, and the preliminary coding. The final table in Appendix 3 (on CD) gathers the codes together under tentative categories, arranged by artist and technologist.

At first analysis, the categories appear to be arranged around three intersecting central issues: communication (e.g. trust, feeling unthreatened), problem-solving (e.g. expressing the problem as an artist and/or as a technologist, compromising), and work practice (who directs work, relationship with technology). In the forthcoming iterations we will see these issues gain richness.

COSTART 2 Reports

The COSTART 2 project is similar to the COSTART 1 project, but collaborators were matched according to lessons learnt in the COSTART 1 project. The corpus of written material consists of some 75

documents of varying types, including project documentation such as artist proposals and case reports, but most interestingly transcripts of interviews with artists and meetings of technologists involved in the project. All documents were examined, but some did not deal with the collaboration process or with artistic relationships with technology, so had no codes applied.

Appendix 4 (on CD) contains the Document Coding Report and the Node Coding Report generated by NVivo software, and collated by hand.

Each document was coded by assigning one or more 'nodes' to passages in the document. Nodes are here treated the same as grounded theory's 'codes'. An example of a node/code is 'Artist–need to change–Technology'.

The Document Coding Report contains all of the codes (48 in total) in all of the documents that were coded (13 in total), arranged in alphabetical order by document name. The Node Coding Report contains all of the nodes (14 in total) that had 3 or more passages coded with that node, arranged by number of passages (highest first). The number of passages that have a code is only an approximate indicator of the importance of the code, just as project documentation is only an approximation of the project itself.

For the sake of clarity, the discussion here will omit codes that reinforce categories that emerged from COSTART 1, but will deal with codes that refine or contradict these categories.

The most common NVivo code, with 12 explicit occurrences, was "Programmer–active interpreter–artist"—that is, that when the artist gives the technologist instructions, the technologist must actively *add* information in order to be able to write code. Often this addition is in the form of the technologist asking neutral questions of the artist, but the answers must still be interpreted into code. Often it is in the form of suggestions or opinions about what should be done. Of these suggestions, a technologist distinguished between "joining-in" suggestions and "technical" suggestions—that is, suggestions motivated by aesthetics, as differentiated from suggestions motivated by technical considerations

(i.e. technological limits, or less work for the technologist). However, the same technologist went on to say that his "joining-in" suggestion was a "behind the scenes technical suggestion" because he "was beginning to realise [he] couldn't do [the alternative]—so a bit sneaky!". An excerpt from a meeting between technologists:

> *sometimes I give a technical reason for not doing something, but later I realise it's cause I didn't like what they wanted to do*

Another:

> *I kind of felt quite worried that I was showing him all sorts of stuff that he didn't know about and that's influencing, changing the way he was thinking about this project. But I wasn't sure that was the right thing for me to do.*

The project documentation is littered with similar examples of technologists' suggestions and conflicting motivations behind them. Of course, this brings into play issues of trust between collaborators ('Artist–trust–programmer' is itself a code with 2 passages), and the issue of how much technologists are aware of the motivations behind what they themselves suggest ('Programmer passive-aggressive interpreter' also has 2 passages). Whilst the technologist may not have made the suggestion had it not been artistically appropriate, it is difficult for anyone to say to what extent technical motivation inspired the suggestion. One of the artists described feeling as though she was at technologists' 'mercy', because of them having decided the 'best and easiest way' of doing the project. Another artist responds more actively to technologist suggestions:

> *I usually think that if somebody says something they have a good reason to say it … of course, if I disagree…*

We will leave aside the second most common NVivo code for a moment, since the third most common NVivo code, with 9 occurrences overlaps the first somewhat: "Technology–inspires–Artist—or—Technology–affects–Art". So as well as the technologist's suggestions or interpretations affecting the technological interpretation of the art, the technology itself will affect the art. A general way of phrasing it, as one

artist did, is that "this is an example of the process creating the aesthetic." The artist was "totally surprised by the outcome of the process".

More specifically, limitations of technology changed artists' attentions (from mobile phone technology to PDA technology in one example), or were exploited as features of the work. The technological and aesthetic result could be similar to the result caused by a technologist's suggestion, except that the influence of technology is not necessarily communicated to the artist via a third party, but is rather more directly apparent.

The ninth code is also closely related: "Programmer–inspiration–artist", with 4 occurrences. Aside from the technologist being an active interpreter of the artist's instructions, there is a reciprocal process where the technologist affects the instructions that the artist gives, or is able to give. This is a strong issue in the COSTART 1 report, and is not developed particularly here.

We can use these NVivo codes to enrich the technologist "Suggesting" category from COSTART 1, with a variety of dimensions of motivations for suggestions (easy vs. hard, active vs. passive (or with passive-aggressive), etc.). In the artist "Relating to Technology" section, we can add "inspiration by technology".

The second most common NVivo code, with 10 explicit occurrences was "Artist–not–programmer" overlaps greatly with the 5th most common NVivo code "Artist–conception–Programming" and the 12th most common, "Programming–too specific for–Art". The first perhaps vague-sounding code refers to instances when artists differentiate what it is that they do with what it is that a technologist does ("I'm not a programmer") or, more occasionally, when a technologist makes that same distinction ("[The artist] hasn't got yet a very clear idea of the logic to use"). The second code refers to passages about artists' conceptions of what programming involves, and the third refers to a feeling expressed by one artist that the programming process is somehow a 'reduction' of the art ("I got so bogged down in all the science part of it, I think I woke up one morning and went 'Is this art?'"). A potentially important point here seems to be artists' conceptions of themselves as "architects"

of a project, and others as "craftsmen" supplying services to the artists' specifications—a key quote in this regard, from an artist:

> *I mean an architect doesn't do the brick laying do they, you know it would be stupid in fact for an architect to build their own houses because their skill is in design, and that's a very important and useful skill, without it the houses wouldn't exist […] my understanding of computer programming is that it's done according to specifications and a bricklayer puts their own flair into laying bricks and for example there are many ways of laying bricks so that they look a particular way […] I just don't want to learn how mobile phones work […] just make it do its thing.*

Tied in with the code about programming being too specific for art is the process of switching thought from art-level to programming-level, and back—from architecture to brickwork and back. There are echoes of my first study here, regarding the supporting the ability to transition between gentle and deep levels of thought, but richness has yet to emerge in this study.

In the COSTART 2 study, only one artist had programmed seriously before, in C then (Adobe Director's[1]) Lingo. Of C the artist says:

> *I used to program in C but I wasn't certainly good at C programming. Of course C is powerful, very efficient but it's also source for problems if you want to avoid headaches it's better you don't use it*

whereas about Lingo

> *high level languages […] make things so easy to do and so that's very important for testing, experimenting […] I'm still not a programmer but I can do things with Lingo and that's good enough.*

Of course, the experience of a single person by itself is nothing to base generalisations on, but it highlights the necessity to consider programming language interfaces when assessing artists' experiences of programming.

---

1 http://www.adobe.com/products/director/

Finally in this area, an interesting example arose of an artist's perhaps naïve perception of the capabilities of computers. As part of the artwork's development, images were to be pre-rendered of various genetic combinations. When asked how many images were to be generated, the artist replied "22 to the power of 21 [...] if we get one or two working then we know the pathways, I think the rest is really possible". At 100KB per image, these images would take 150 billion billion gigabytes to store, and the artist appears not to have realised the impossibility of such an endeavour. It is unclear whether the calculation is incorrect, or whether the technologist has missed the computing implications, or whether the "one or two pathways" were meant to be a pre-calculated prototype for a future on-the-fly calculation of genetic combinations.

These NVivo codes enrich the COSTART 1 categories of the artist "Working with Technologist" (now renamed "Relating with Technologist") and "Relating to Programming". The artist-as-architect, programmer-as-bricklayer analogy, for example, describes the delegation of work from the artist to the technologist, and highlights the issue of the scope given the technologist to add individual flair (as opposed to constructing the program only according to specifications). The other side of this coin is the extent to which the artist needs to be interested in the programming process—can it remain 'a world of mystery', or does the artist need to understand something of the difficulties the technologist faces, or the program's, 'layers', or structure? The fourth most common NVivo code, "Programmer–engagement–Artist" marked 6 passages, all in the minutes of a meeting of technologists, where the topic was raised. Technologists appeared to vary in their engagement with the artwork that they were collaborating on, and also in whether their engagement was 'technical' or 'aesthetic'. Overall, interesting technical problems appeared to motivate the technologists more than interesting artistic problems (e.g. "even if I didn't care [presumably artistically] that much I would still try and do it in the best way you could do it and go back to things and rewrite it"), but the distinction was not always clear in the documented opinions. For example, one technologist states,

> *you do the best you can but it's sort of flat, whereas if you're involved with the artist in particular or the challenge intellectually then you be more inventive, creating more solutions.*

Other possible motivations mentioned were money and ownership. One technologist said,

> *perhaps you're being paid to write [the] program rather than caring about the work, [with] the result [that] you don't feel so engaged as a person, you don't want to own it.*

This ties in closely with the issue of the technologist's influence on the art—not only the motivation to make suggestions, but also the motivation to be involved in the collaboration in the first place. The new category "Motivation" was added for the technologist.

The sixth NVivo code is "Artist–need to change–Technology", which describes occurrences in the project where an artist expressed a desire to change technology in order to accomplish his or her vision. As discussed in the first study, these are cases of the failure of the constraints of a medium to keep pace with a creative person's changing conceptual spaces. There are 5 passages coded with this code. Two examples:

> *[Interviewer]: So you couldn't actually manipulate the image to your satisfaction?*
> *[Artist]: No, that's right, not at all. I think that that's the frustrating, in fact that's been the frustration throughout my whole art practice.*

and,

> *"If Technology couldn't develop as I wanted, I have to change technology itself."*

This reinforces codes in COSTART I, such as finding technology limiting, finding programming constraining, compromising quality because of technology limitations. The earlier code of 'features drawn from limits of technology' is also reinforced. However I think the point can be made more strongly in the code scheme: the artist requires, if necessary, full control over a system, and perceives technology as being too rigid.

The seventh NVivo code, with 5 passages, is "Problem Solving/Analysis vs. Synthesis". This code relates to the variation in problem-solving approaches used in the collaborations. The 12th most common code, "Problem Solving–compromise", with 3 passages, is related. Some described approaches attempted to 'plan' the problem-solving (synthesis), and others attempted to 'discover' solutions by 'messing around' and seeing what happens (analysis). There seems to be a perception amongst individuals that every person prefers one or the other, and there also seems to be a tendency for technologists to 'synthesise' solutions and artists to produce solutions by 'analysing' (computational) situations—although exceptions and concessions on both sides were common. In both approaches, where described in the documents, consideration of the available tools seemed to play a significant part in organising the problem-solving process. One technologist drew out an interesting point about the Max programming language:

> *I think it's good but […] I think it, there are some problems with it. One is that your [sic] encouraged to fiddle which I don't like very much. I think it seems to me so unlikely that you'd ever arrive at something, well first of all it seems to be unlikely that you'll ever arrive at something good by just fiddling around. […]I want to have a clear idea of exactly what it is I want to do because otherwise you don't understand the problem and science seems to be about understanding the problem. But it is, on the other hand there are some advantages to fiddling like that.*

So even though several artists have described Max as a useful tool for playfully understanding and exploring problem spaces, this particular technologist preferred a more considered approach to problem-solving (at least in this collaboration). This enriches the "Expressing problem" (renamed "Solving Problem") category—a possible tension between "fiddling" and problem-solving, or between analysis and synthesis approaches for thinking about computation.

As for "Problem Solving-compromise", this code marks occurrences were artists described trade-offs between different potential solutions. To take one example, an artist considers 3D software that can respond to sound but not change shapes of items, against 3D software that can

change shape of items, but not in response to sound. This is another example of an artist's conceptual space lying outside the constraints of available software, but not outside computing fundamentally (see Study 1 (Chapter 5) for a fuller discussion of the consequences).

Compromises are not always so dichotomous. In several examples, an iterative approach is used to navigate a path that balances two alternative approaches to problem solving. As an artist describes it:

> *There are authors who prefer an approach that is completely free but also quite arbitrary, and others who on the contrary prefer to follow precise rules. As always only the results are important, as they tell us if a certain approach actually works. I believe that a way in the middle is the best solution.*

Other, less common codes, also deal with different problem-solving approaches. The 14th most common code is "Problem Solving-interesting approach", which denotes a variety of responses to problems, including question-asking, exploiting the features of the problem, and becoming emotionally-involved in the problem.

The eighth most common code is "Visual Programming Environment–convey structure". The documents rarely focussed on of the actual programming process, so analysis of particular language features is difficult to make. However, in two documents the qualities of Max/MSP were discussed, particularly with regard to its conveyance of a program's structure. Overall, despite being a visual programming language which by nature is a graphical representation of the program, it seemed that technologists had to spend a great deal of time arranging the visual elements so that the organisation makes visual sense, often to the extent of, apparently completely re-creating the program: "So like with [Artist] we did it and then before the exhibition I had to go through and did it again so it was actually crystal clear what was happening and that's not necessary its just something I do." Creating and conveying structure appears to be an aid to the problem-solving process for the technologist, and to understanding the programming for the artist—'structure' codes are added to these categories.

The tenth NVivo code deals with the artist's expectations of the artistic environment—in other words, instances where the artist describes arriving at the point where they knew what to expect of the environment of the artwork's creation.

> *If it's a world of mystery for you does it mean you have expectations of it that are unrealistic or perhaps… not unrealistic but at least difficult for you to actually know whether something that you want done can be achieved,*

writes one artist.

Interestingly, the other two occurrences mark the situation described before, where an artist became confident that "the piece is actually possible now" after calculating how many permutations of images would have to be pre-rendered. The confidence was misplaced, since the number of calculations was actually vastly beyond the capabilities of modern computers.

An interesting issue is raised here: In a collaborative art process, where an artist may well know little about some aspects of the artwork's development, it can be easy to misplace confidence in, or have false expectations of these aspects. In this example, knowing 'an answer' existed was sufficient to give an artist confidence in the approach, but the computational implications of that answer were not available to her.

This state undermines the perception held elsewhere (in COSTART 1) that "knowing the rules means that the solution can be found". Sometimes knowing enough rules means that it becomes clear that no solution is possible within those rules.

The 11th code is related: "Computer–obscure to–Artist". This relates to one specific artist's frustration with the computer support in her collaboration. This is the same artist who made the architect-bricklayer analogy, but later in her interview, she expressed a need to understand at a high level about what the computer was doing, and what it potentially could do:

> *Artist: No, well I've never directly asked anybody but I find that the whole process actually of dealing with [?] totally frustrating because*

> *[SysAdmin] is not interested and I think he has his own logic which doesn't fit into mine, so I don't think he really cares about trying to tell you either how it works, that's not his job.*
> *Interviewer: I'm just interested in why it matters to know as long as… the application…*
> *Artist: Because… then I don't need help, cause I hate asking for help and so if I…*

and later:

> *like if you don't know what you've got you can't use it, can you?*

These comments were in relation to the applications available on a given machine, and her access to them (as granted/introduced by the SysAdmin) rather than the possibilities of an individual application or programming language.

This is an isolated case, but it reinforces the variety of constraints that artists may need to overcome. The artist maintains no particular interest in programming, but still needs to be in control, and aware of the possibilities of the computer, at the level she is interested in working at.

The remaining codes each had two or one occurrence(s), which I will treat as being too isolated to be worthy of mention in this analysis. However, several of these codes can be incorporated into one of the more 'saturated' codes. For example, the code "Programmer professionalism" can supplement the technologist's motivation category, and the communication with artist category.

The "Artist–paranoia–Computer" code can be seen as a manifestation of the artist's relationship with the computer, and the 'world of mystery' that programming is sometimes seen as.

I have also removed some codes from COSTART 1, as they received no particular treatment in the COSTART 2 report, and don't appear to be especially linked to the rest of the coding system. These codes are, on the artist side "Retaining Creativity" and "Maintaining Quality" (these appear to be always important to the artist), and on the technologist side "Predicting" and "Revising Estimates" (whilst these are undeniably part of a technologist's role in managing the expectations of the artist, they

appear to be more a consequence of project planning and resource limitations rather than of a technical or creative nature). So, incorporating the codes from the COSTART 2 documentation into the COSTART 1 open coding table from Appendix 3 (on CD), has produced an updated code table, still arranged by artist and technologist. This updated table appears at the end of Appendix 4 (on CD).

Summary

The three intersecting central issues found in the COSTART coding table—interdisciplinary communication, problem-solving and work practice—remain, and have each gained supporting codes from the COSTART 2. A tentative analysis indicates that the codes within each issue fit well with established knowledge. For instance, communication appears to be contingent on, amongst other things, the well-known issues of trust (and honesty), shared language, agreed roles, question-asking, and whether question-asking is encouraged. Work practice is contingent on, amongst other things, domain knowledge and different social settings (working alone, in close partnership, with others). Problem-solving approaches used several well-known creative approaches—analytical vs. synthetical approaches, intuition, compromise, etc. None of this is particularly surprising. However, the *intersections* of these issues may be of particular importance in creative collaborations. For instance, the motivations of technologists' suggestions emerged as a category with several supporting codes in the COSTART projects, and arises between technologists' work practice and their problem-solving approaches, and manifests in communication with the artist.

To take another example, the codes relating to the artist's relationship with technology and with programming, and the technologist's active interpretation of the artist's views, and the sometimes formal separation of artist and technologist work practices indicate that there may a conceptual gap in artists understanding of computing technology, and that technologists are responsible for much of the mediation across this gap. They are interesting in the light of Study 1, but suggest little in the way of remedy, and besides are rather unsaturated in the COSTART documentation. This is to be expected for three reasons: firstly, the focus of the study was on supporting collaboration, or the computer *as mediated*

by the technologist; secondly, the short time and high pressure of residencies left little time for paradigmatic learning on the artist's part; thirdly, the COSTART interviewers were non-programmers.

The next stage is to gather, code and analyse new evidence in order to add richness and gain confidence in the theory. The second part of this study, in the next chapter, describes that stage.

7. Study 2, Part B—Technologists' Roles in Interactive Art Collaborations: Interviews

Introduction

This chapter is the second part of the grounded theory study begun in the last chapter. The coding of the COSTART project data in that chapter has raised a number of interesting issues, and it is time to gather new evidence to explore these issues. As described and justified in the methodology, the new evidence takes the form of interviews with artists and technologists about the issues raised by the coding so far. In this

chapter I will describe the rationale for the particular interview questions asked. Having coded the interview transcripts (which appear, with editing for confidentiality, in Appendix 6 (on CD)), the coding was incorporated into the coding table produced by the COSTART coding, to produce Appendix 7, the complete coding table, with categories in a hierarchical arrangement, quotes from the interviews, and memos to assist my drawing the categories together into the theory. At this point, enough categories were saturated to form the super-categories: Relating to the Project, Collaboration Patterns, Developing Problems into Shared Structures, Artists Exploring Technological Structures: Naïve Interactive Art, and Human Computational Interfaces, Technologists Exploring Artistic Structures: Intimate Iteration and Computational Toy-Making, and the central category, Attuning. The write-up of that table, incorporating quotes and memos is the analysis contained in this chapter.

Interview Design Rationale

The question list can be drawn from popular, yet unsaturated codes and the accompanying discussion, with some additional questions to establish the individual's context. I present here the design rationale for the interview questions. The formal list is presented in Appendix 5, and the methodology chapter (especially pp. 114-115) deals with techniques for appropriate question wording, but the actual interviews were semistructured, so often diverged from this list as I attempted to draw out interesting topics. The transcripts of the interviews are in Appendix 6 (on CD).

The questions in Appendix 5 will be referred to in the following discussion by the notation AQ 3a, which means Artist's Question 3, part a, or TQ 5b, which means Technologist's Question 5, part b. Many questions for artists are mirrored in the questions for technologists (for example, the background/history questions, and the computer use history). Whilst not reflecting similar correspondences in the coding, I have added these mirrored questions in order to be better able to compare the experiences of artists and technologists.

GENERAL QUESTIONS

The general questions are to provide a general indication of the respondents' general artistic history (AQ 1a) and experience with computers (AQ 1c; TQ 1a, 1b, 1d), and experience making art with computers (AQ 1d, 2 and 2b; TQ 2 and 2a). Artists were asked more closely about their art experience; technologists were asked more closely about their computing experience.

MOTIVATION (MACRO AND MICRO)

The issue of motivation emerged as a strong feature of the COSTART coding.

By macro-motivation, I mean motivation for the collaborators to work in their field, and in interactive art collaborations—what drew the collaborators to work together, and what each person wants to gain from the collaboration. wanted to compare reasons for making art (aesthetic motivation) with reasons for using computers (technical motivation) Hence AQs 1b, 1e and 1f deal with an artist's reasons for making art (with computers); AQ 2d asks the artist to reflect on liked or disliked collaborations, AQ 2e asks about the preferred power relationship in the collaboration (with reference to Mamykina's and Candy's treatment of the same (Mamykina, Candy, & Edmonds, 2002)); AQs 3b and 3c attempt to establish the artist's interest and involvement in what the technologist does. TQs 1a and 1c deal with the technologist's reasons for programming, TQ 2b, 2c and 2d are questions about the technologist's likes and dislikes in collaborations, motivations, and observed power relationship (again with reference to the Mamykina/Candy paper).

By micro-motivation I mean the motivations guiding collaborators' individual decisions—particularly the technologist's, since these have emerged as important in the coding so far. Hence AQs 3, 4, 4a, 4b offer the opportunity for the artist to talk about potential difficulties in collaborations, and the impact of the technologist's influence. TQs 4, 4e, 5, 5a, 5b provides a similar opportunity to technologists.

THE COLLABORATION PROCESS

AQs 2a and 2c attempt to establish some initiation criteria for the collaboration—what makes a non-programming artist decide to make art

that requires programming. AQ 2e follows on from this by asking what style of collaboration the artist prefers—what expectations the artist has of a technologist, and why.

AQ 3a is the beginning of an open-ended discussion on the nature of the collaboration process. This may vary significantly from respondent to respondent, and from collaboration to collaboration, so I have kept the question general, without mentioning specific aspects of the process. If more detail is required, it can be asked at a later date.

Finally for the artist, I have included AQ 3d in order to gain an idea of the completion criteria for a collaboration. In other words, perhaps the artist may have a very specific idea of what the artwork should look like; or perhaps the idea is more vague. If the idea is vague, I am interested to find out the process by which the artist can make a vague idea more specific, and whether the computer or technologist plays a role in that.

The technologist is asked corresponding questions about his side of the programming process. The collaboration context is set in TQ 2d as well as the general and motivation questions. Question TQ 4a asks the technologist about selecting appropriate tools, with the intent to discover what factors are seen to be important, and TQ 4b asks the technologist about the process of development with those tools. TQ 4h attempts to get the technologist to relate art development to 'conventional' software development.

I also decided to ask the technologist about his interactions with other technologists (TQ 3a and 3b), since the topic came up a couple of times in the COSTART documents; and some artistic collaborations may involve multiple technologists or technologists.

ARTISTS' AND TECHNOLOGISTS' RELATIONSHIPS TO COMPUTING

In the light of the lack of positive data about Artists' conceptions of computing, I have decided to include some open-ended questions and points of discussion, to attempt to saturate codes about the artist's conceptions of computing, and to provide clues about potential remedies to the problems described. TQ 7 is the clearest example of this, and is

based upon the findings from the first study about the importance of object orientation. It seems meaningless to ask non-programming artists about what might improve their own ability to program, since they are unlikely to have the necessary perspective to tell what knowledge they lack, so I asked artists more general questions than of technologists about the artists' relationships with computing. These are AQ 1g (which asks about artists' understandings of computers), AQ 1h (which asks about artist's opinions of their own programming), AQ 3b and 3c (which explore the artist's motivation to get involved in the programming progress, and seeks ideas for making involvement easier). To the technologist, I asked more specific questions, starting with TQs 4c and 4d, which attempt to gain information about what happens when the technologist 'lets the artist into their world'. TQ 4c is designed to elicit responses about sharing technological problems with the artist, but I use the word 'show', rather than 'describe' to prompt technologists to consider circumstances when they used the computer to assist. TQ 4d is more specific dealing with how the technologist managed showing programming code to the artist, and what the technologist thought the artist's response was. TQ 6 takes the topic in a different direction, and deals with the possibilities to convey abstract principles of computing. TQ 4e shifts the technologists' focus to a more general consideration of the artists' perceptions of programming.

TQ 4f and 4g ask the technologist for ideas to help the artist engage with the technologist's work, and for ideas for computational support for communication. The questions ask for the technologist's own solutions, and for his 'wish-list' for technology.

Interview Analysis

I interviewed four technologists [IT1–IT4] and two artists [IA1–IA2] for this stage, who were the respondents to a word-of-mouth call for respondents in the Sydney interactive arts community to non-programming artists, and technologists who had programmed for non-programming artists. The annotated transcripts of the interviews are in Appendix 6 (on CD).

The respondent for the first interview [IT1] participated in the CO-START projects as a technologist, so I asked him further questions

about his COSTART interviews. He and I had also recently worked on an art project together as technologists for a third party.

Of the other respondents, the two artists and one technologist [IT4] were, at the time of interview, working on a large project together. Their interviews are the final three of the appendix. I observed a production meeting for the project, and noted with interest that the technologist had created a graphical interface for manipulating parameters of an algorithm, which was a control element of the artwork. I decided to ask questions about the technologist's rationale for making the interface, and the artists' reactions to it.

With regards to the selection of artists, there is scope in future work to study a greater number and wider range of artists, but a) the focus of this study is on the roles that technologists play in collaborations and their effect they have on artists' ability to do computing in collaborations and b) as I noted in the COSTART analysis, artists speaking from a position of ignorance about computing (in terms of deep programming) are unable to illuminate the process a great deal. These artists are included in order to provide multiple perspectives on the particular project they were collaborating on with [IT4], turning that project into the central case in the grounded theory. This may introduce a bias towards the views expressed by the interviewed artists, but the strong consistency of categories that has nevertheless emerged between the concerns of the interviewed artists, the COSTART artists and the combined technologists is evidence that this hasn't been the case.

The interviews were selectively coded, by hand, and the codes were incorporated into the open code table produced from the COSTART analysis. I removed unsaturated categories, and reorganised the table until a hierarchy of categories emerged (Appendix 7 has the coding table, and a detailed description of the process undergone to get from the COSTART coding to the final arrangement of categories).

The resulting coding table is shown in Appendix 7 part 2. The majority of memos from that table are written up to form the theory that follows. The memos relating to technologists' relationships with technology are covered more appropriately in Chapter 8, design recommendations.

DEMOGRAPHICS

All respondents were native English-speakers. All were Australian except for one technologist (British, living in Australia).

The technologists had varied professional backgrounds. All were male. Three had done degrees in computer science or electrical engineering, whereas one had done a (first) degree in product design. They all had many years of experience in programming, but only one had voluntarily programmed since childhood. Their ages ranged from 28–36 years (as established subsequently to the interviews). They had varying experience in programming art projects. [IT2] had least experience, with two public artworks, and [IT3] and [IT4] appeared to have most.

The artists were more similar. IA1 was female, IA2 was male, and each had made artworks for more than twenty years. Both had come to interactive art from film-making, but IA1 had an education in media production, and IA2 had an education in history.

There are two things to note here. Firstly, in comparison with the COSTART participants, these respondents are significantly more experienced art-technology collaborators. Secondly, the participants had all chosen for themselves who they would be collaborating with, whereas in COSTART this was decided by group consensus, since resources had to be shared amongst many collaborations. These things sensibly explain why some of the categories that emerged from the COSTART data were not saturated by the interview data, and so did not emerge into the theory (for a list of which categories, see Appendix 7, section 1). I will emphasise that I have consciously set up a false dichotomy. Technologists and artists are not always mutually exclusive; I have separated them here so that we can get a better understanding for the concerns guiding each way of being.

ATTUNING

The theory holds that of fundamental importance in art-technology collaborations is the process of 'attuning' around the artist. The attuning, like the tuning of an orchestra around a 440Hz A, takes place between the collaborators, as they develop a shared understanding of what is meant by particular concepts, words or artefacts. As [IA2] puts it:

> *the hardest thing about working in any multicultural situation … is finding some sort of communication system where everyone has at least 70% chance of referring to the same thing … and maintaining a good shared memory of that language and of decisions made*
>
> [IA2]

Like the tuning of one instrument, each individual attunes the shared understanding of these concepts, words or artefacts with their individual understandings. Hypothetically, a technologist may see a picture on screen as the result of a rendering operation based upon certain parameters, whereas an artist may see the same picture as the representation of an emotional or conceptual process.

Collaborators must also attune to their place in the collaboration, the motivations for collaborating, and the points at which each person's expertise and responsibility comes to the fore. Like the tuning of a car engine, collaborators work best when they are well tuned. And also like a car engine (a COSTART technologist made the direct comparison), the computing system works best when it is attuned to the needs of the people using it, and it is to a large extent the technologist's purpose to enable that tuning.

The following sections will explore the various aspects of attuning in a less metaphorical sense: the relationships of artists and technologists with the project, with the problems to solve and with technology. Throughout each of these permeate the relationships of artists and technologists with each other.

RELATING TO THE PROJECT

Technologists are motivated to work on interactive art for several stated reasons, chief amongst which was the enjoyment of the challenge. One technologist specifically stated:

> *I was more excited about doing the things I didn't know how to do.*
>
> [IT3]

Artists appear to value this ability somewhat ("broad range of intellectual and personal experience" … "not threatened by reaching out past

what they know about" [IA2]) though they also value experience in a technology, if it's been identified:

> *choosing a programmer is based on the environment I think it's going to be developed in and the advice I get from the programmer about the appropriate environment.*
>
> [IA1][1]

> *I would be looking at [whether] they'd done similar work before*
>
> [IA1]

Even though [IT3] is more attracted to things he doesn't know how to do, he has also gained familiarity with many art projects (not just his own) and readily exploits that as a problem-solving technique (see relating to problems, later).

Technologists also work on art because they see it as "soulful" [IT3] and "cool" [IT1]. [IT3] phrases it as "a sense of expression beyond simply […] processing". There is a possibility that the combination of the challenge and soulfulness of the project is what sparks the technologists' excitement, creativity and interest/curiosity/engagement (valued by artists), other reasons given for collaborating on art.

More generally, technologists are attracted to computing as a medium for several reasons. Every technologist interviewed was motivated by the satisfaction gained by solving problems or achieving a goal, and frustrated by the inability to solve problems, or the lack of a goal. This attraction to completion is reflected in the technologists' levels of satisfaction from art projects:

> *if someone sets me out a goal that I find hard to nail down … [it's] going to make me unsatisfied*
>
> [IT3]

1 this quote suggests a possible Catch-22 dilemma. If a programmer suggests the environment, why is the choice of environment used to choose a programmer? This could be a case of an artist making premature technical decisions (as described by one programmer)

> *well I did enjoy the [Artist name] one, but it was annoying that we hadn't got it finished*
>
> [IT1]

[IT1] compared programming to writing, and [IT3] related it to philosophy. Both characterised these other creative activities as more open-ended:

> *I suppose you get "flow" [in Czikszentmihalyi's sense] or one of those things [from programming], which I don't get with writing, or not very often, anyway.*
>
> [IT1]

> *I found philosophy to be the opposite where you cant just plug away and get to a position…*
>
> [IT3]

Technologists often used the terms "satisfaction" and "enjoyment" when describing the feelings that their motivations provoked. One technologist [IT4] distinguished between the two, saying that enjoyment took place throughout the collaboration, and satisfaction required 'the final form'.

[IT4] described many more of his attractions to programming in this rather eloquent paragraph:

> *I have a general interest in building things. That is a process that I very much enjoy and I've got a number of different media for it. Software obviously has lots of interesting things about it in terms of its lack of resource limitations, beyond simply the process of imagination and design2, and that's one of the things that's really extraordinary about it. The elegance that you can achieve in that domain, even if it's not necessarily always as visible as it might be with a more material design and construction form. It's extraordinary. That's something that has interest and beauty and all sorts of other potentials that are curious and fascinating.*
>
> [IT4]

2 This is a product of "Servitude", one of my four Ss of computing (see p. 146–149)

The artists I interviewed work in art because they are 'compelled to', and in art collaborations because they have a vision for a project that requires expertise that they don't have the time or interest to gain (this will be explored later, in relationships with technology). Artists value agility/nimbleness in technologists, and devalue inflexibility.

> *In a programmer I look for linguistic and conceptual nimbleness and agility"*
>
> [IA1]

> *he turned out to be very inflexible and grumpy […] wouldn't meet us part way and that was turned into a bit of a nightmare"*
>
> [IA1]

I would argue that this is related to the flexibility required of all creativity support systems—to shift constraints as a creative person's conceptual space changes. This flexibility is also desired of other humans included in the system who would also need to adjust their constraints to respond to this change. I suggest that this agility is necessary to 'attune' effectively, in all of the various ways that the technologist is required to attune, and the very word is reflected in the 'agile programming' methodologies that the technologists regularly employ.

There was also a value attached by artists to the education of a computer technologist:

> *[I'd consider someone who's done] computer science and engineering and who can program C a proper programmer" … "I chose someone who was essentially an arts student dabbling […] who wasn't a real programmer so really that's why I've got quite firm about [quality of programmers]*
>
> [IA1]

[IA1] later elaborates:

> *I thought no I want a proper programmer. Because clearly it's a whole way of thinking. And a whole way of conceptualising […] you know it's a very respectful position. I've never had a programmer who hasn't shown respect in my area of expertise, and I hope I've always*

> *respected their area of expertise. So in that way I think it is an equal thing.*
>
> [IA1]

This seems to imply that the technologist is respected for his different way of looking at things as much as for his expertise with computers.

Artists also value 'people skills' in technologists. This is made explicit perhaps because of the common reputation of computer scientists (and creative people in general (see pp. 32-34 of this thesis)) as possessing antisocial traits. In particular "goodwill", "sensitivity" [IA2] and "how that person will work with the rest of the team" [IA1] and "grumpiness" and "arrogance" [IA1] are devalued.

Artists also value programmers who are quick and cheap, though this is likely to be for pragmatic (and commercial) purposes more than for creativity support purposes.

COLLABORATION PATTERNS

Mamykina and Candy used the COSTART data to discern types of artist–technologist collaboration (technologist as assistant, full partnership with artist control, and full partnership). However, my interview question that related those findings to my respondents produced some interesting answers, which adds richness to Mamykina et al.'s theory. Two technologists, [IT1] and [IT4] mentioned examples of equal partnership collaborations, but otherwise technologists tended to characterise themselves as consultants and problem solvers, and artists tended to characterise technologists as craftspersons and consultants. There is a definite sense of delegation from the artist to the technologist.

[IT4] also raised a fourth possibility—the role of technologist as teacher. Moreover [IT4] is motivated to collaborate by the possibility to teach.

But the most interesting finding to emerge is that the hierarchy between artist and technologist rarely falls neatly into one of Mamykina et al.'s modes of collaborating. There are many examples of individual leadership and decision-making in the data. As [IA2] puts it:

> *I wouldn't prioritise one [mode] over the other but I do find that it is a dynamic situation. There is always a regime in which a project is being made and that's usually a funding and or a research regime and there is usually someone for whom the buck stops who's most exposed in a way by the whole project so that is usually the person with responsibility and authority and there's no denying that […] [T]here are moments in the narrative of delivery of the project when certain people's expertise, responsibility and energy and enjoyment needs to come to the fore and they are the driving force for awhile. Which doesn't necessarily mean that the person or people who have been vested with authority let go of their authority, but there's a dynamic system in a good collaborative enterprise where certain people responsibility comes to the fore at the right moments and everyone needs to be nimble enough to allow that to happen.*
>
> [IA2]

In short, the person "for whom the buck stops" (who in COSTART was neither the artist or technologist but the research leader) is always in overall charge, but, as a consequence of the nimbleness and agility required of the technologist (and presumably other attuning collaborators), there is a constant shifting of authority and responsibility amongst the collaborators. This is reflected in particular examples from the interviews:

> *if they want you to do something and you tell them that it's not going to work and it's your job to know, then they'll believe you, and they'll ask you what will work*
>
> [IT3]

> *The model that I have found myself in is—to me, it's a difficult divide. I've always tried to be very conscious of the implicit and unspoken aspects of this sort of collaboration, the way in which people's confidences and competence and sort of notions of demarcation can actually become entrenched without actually being specifically discussed […] I really try and contest these boundaries and limitations between myself and other people I'm involved with*

> *as much as I can. I try and bring as much as I can aesthetically and creatively to a project as well as try and keep the technical side of things as permeable as possible as well. The project belongs to the artist and the artist is therefore in the position who you'd expect to be able to schedule time frames, and lay the work out but it comes down to the programmer to do the work and give expectations or estimations of time […] the artist depends entirely on the competence of programmer in every sense and also their ability to express and estimate these things. But the actual operating and running it and keeping things moving and everything else generally falls into the hands of the artist and that can be a contested relationship sometimes.*
>
> [IT4]

What one artist calls a "dynamic system" a technologist calls a "contested relationship". The aggressiveness of the latter term is possibly as a consequence of whether everyone is nimble or entrenched, or whether the dynamic nature of the roles and consequent responsibilities are made explicit or not. [IT4] mentions as a general example situations in which inappropriate technical decisions have been made implicitly (without consulting him). A few different production methodologies were described. Most technologists, and one artist, described agile programming methodologies (for example, extreme programming) as being preferred. [IT2] (in art) and [IT3] and [IT4] (for commercial work, or for taught courses) had developed software using a more traditional software engineering process (with a detailed requirements phase and technical documentation). [IA2] described his dislike of such processes:

> *I enjoy less, and I find less productive a kind of engineering model of working where a brief gets written and instructed down the line to the programmer […] occasionally because of the institutional frameworks that I've found myself in I've been compelled to work that way […] I find that it meets certain pre-determined schedules but it delivers less than satisfactory results*
>
> [IA2]

[IA1] compared an 'art school' methodology with a 'media production' methodology:

> *I think there is definitely a school of artists who have an attitude if you don't do it all yourself its not art. I have a background in media production and when I did my undergraduate here in film its like if you wanted to work alone what's wrong with you and individualistic romantic shit.*
>
> [IA1]

However, this media production process (coming as it does from film production) is somewhat similar to traditional software engineering, in that it requires a great deal of up-front conceptualisation and planning:

> *your concept for a film needed to have theoretical motivation, conscious aesthetics and a lot of planning … I came from a position of being able to strongly conceptualise things*
>
> [IA1]

Clearly this is at odds with the agility (and need to experiment, dealt with later) that is so valued by artists, and promised by (particularly experienced) technologists through computers and agile programming methods. [IA1] is aware of this, as:

> *what I'm trying to do in my own artistic development … is learn more intuitive, iterative, work alone playful type practise*[3]
>
> [IA1]

To this end, a "technical producer" was hired for the art project, part of whose role it was to incorporate agile methods and playful development into the planned media production structure. This approach was tricky, as it appears to involve a significant degree of conflict resolution, but had some success as we shall see later.

3 yet I am not sure whether [IA1] intends this playful practise to be afforded to her collaborators, as the [IA1] later says "I can't stand around and wait for people to make … mistakes that 20 years ago I knew how to avoid by planning"

DEVELOPING PROBLEMS INTO SHARED STRUCTURES

In the vast majority cases where it was mentioned, artists presented, or were characterised as initially presenting, what to a technologist sounds like very metaphorical descriptions of the systems they envisaged, as opposed to delivering precise descriptions. These descriptions were variously characterised as 'vague language', describing work 'in terms of effect', and consciously communicating a 'metaphorical understanding':

> *it's about helping them getting a handle on the aims of the work, the kind of effect you want it to have on the audience […] broadly speaking*
>
> [IA1]

> *[I keep] in myself and communicate to the others a metaphorical understanding of the objective … I shy away from a detailed description of the project, but I try to offer again and again quite carefully considered metaphorical accounts of the project, it's like this, it's like this, it's like this" … "signal, signal, signal, transmission, signal until you all start to think ok we've attuned ourselves somehow.*
>
> [IA2]

(A possible exception is [IT2]'s report where "[everyone had] a fairly good idea of the features we needed before we started" [IT2]. However, "these features definitely changed and developed as the art work applications developed" [IT2], which implies that an iterative approach must have been necessary to adapt to the changes.)

There was particular evidence in the COSTART projects of dissatisfaction with this approach amongst technologists, who wanted more specificity, logic and sequencing (perhaps due to their motivation of achieving goals or finding solutions), but this was not reflected in the interviews, and could be due to the projects' technologists' inexperience of working with artists. [IA2]'s repetitive language implies an iterative process of communicating the artistic vision—the 'reflexion' of the mostly iterative process of developing the technology. In fact, several technologists describe their problem-communicating process as being similarly repetitive and active:

> *the first thing that happens when people come to me and ask me for help in doing a digital artwork is that I ask them lots and lots and lots of questions*
>
> [IT3]

> *do you mean this, this and this?" … "take a best guess […] with the knowledge that […] there may be backtracking involved*
>
> [IT4]

Other techniques were to ask what "details" of the work were critical, or to enquire about the outcome of particular scenarios.

The intimate repetition and mutuality of the two-way communication, as well as intimate attention to detail, can be seen as an attempt to resolve ambiguity sufficiently for the technologist to decide how to begin transforming the artistic concept into technological terms, and to find the possible constraints of that system. Such to-ing and fro-ing, shared language development and constraint adjustment, is a clear example of creative attuning.

In response to the high-level approach of the artist, technological development appeared to proceed in a rapidly top-down, then gradually bottom-up fashion. In other words, first a high-level sketch was made of the entire system, then low-level technology tended to be the first to actually be made. As noted in the literature review, in Guindon & Curtis' (1988) study on software designers, they identified an opportunistic strategy where designers move between the problem domain and the solution domain, at high and low levels as they see fit.

[IA2] alludes to this process:

> *the means by which the programmer is both creating that knowledge and acting on that knowledge formed thus far is to specify the suite of needs that the program might have to address in the next stage and therefore through a quite rapid and in some ways messy iterative, iterative, iterative process you watch the project emerge in a way.*
>
> [IA2]

What causes these messy process? Why not proceed in a more uniform manner? [IT3] uses a bodily metaphor in a partial explanation:

> *the hardest thing about programming for me I think is curbing my urge to just start straight away. Because I find the programming part, the actual brains part, the piece of software to be by far the most fun and the hardest thing for me is to not work on that until the skull and exo-skeleton and hair and everything is all ready for the brain to go inside it because invariably you'll make the brain wrongly if you don't have the right shaped skull for it*
>
> [IT3]

Earlier, [IT3] had said:

> *if there's no input or no output then its hard to simulate accurately what's going to go in the middle and its usually input is the hardest thing to manifest output you can fake pretty quickly and that's not a problem but input is often really complicated"*
>
> [IT3]

So we can infer that, in interactive art, [IT3] prefers to build the input and output systems (the 'body') before the bit in the middle ('the brain'). The input is 'hard to manifest' because, in interactive art, it comes from members of the artist's audience, or the artist themselves. In other words, input arrives as a result of human presence or actions, and subject to the vagaries thereof[4].

Also in relation to bottom-up development, [IT4] sees how small parts operate and has to think about how to fit them into a system:

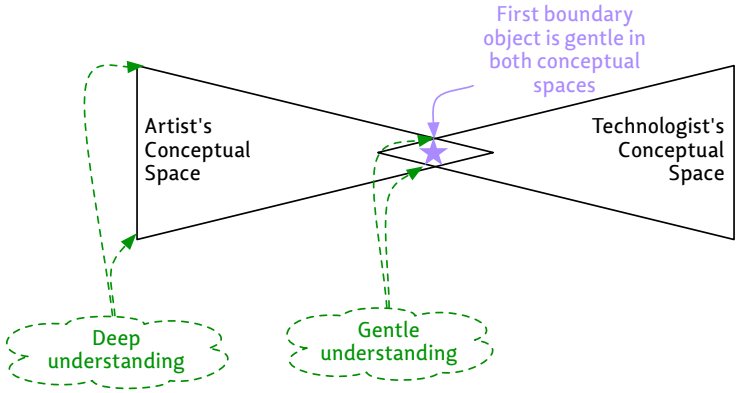
> *top-down is very different from the bottom-up approach that most patching environments encourage … it's quite hard to switch back and forth between those things*
>
> [IT4].

[IT4] makes the point that patching systems (like Max/MSP) encourage a bottom-up approach, which is presumably partly because their inter-

4 Footnote: for an interesting account of such issues see Rokeby (Rokeby, 1996)

faces are predisposed towards building structure starting from simple, concrete rather than complex, abstract elements.

Although there are technological reasons for proceeding in bottom-up fashion, artists still tend to describe the requirements in a high-level way, leading to the erratic development processes that were reported. I would like to propose a different way of thinking about the progression of development in these collaborations, not from bottom to top or top to bottom, but from gentle to deep. To help, I will first hypothesise a wider motivation, based on the attuning process, and with the aid of this sketch:



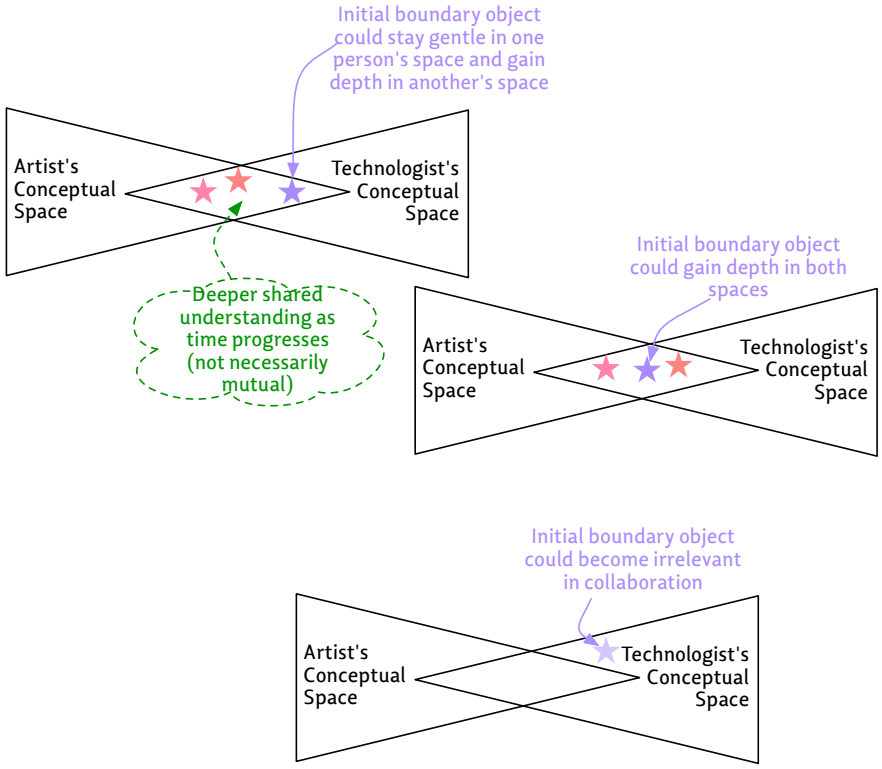
The two triangles represent the conceptual spaces of the artist and the technologist. The artist has 'deep' understandings of the artwork—its history, personal meaning and aesthetic purpose, and 'gentle' understandings of the artwork—what colour it will be, which films will be included. Similarly, the technologist has 'deep' conceptions of programming—the 'grand beauty' of computational abstraction, or an intimate grasp of assembly language, and 'gentle' conceptions of programming—how to change the colour of a shape on screen, or track mouse movement.

(NB. the sketch is not intended to imply that artistic and computational deep and gentle understandings are always diametrically opposed—they could be very greatly aligned, which may make attuning easier. Additionally, the amount by which the technologist gains artistic understanding is not necessarily the same as the amount by which the artist gains technological understanding)

The first technological developments are usually computationally gentle—a sketch of high-level architecture (often on paper) or a computational sketch of a subsystem for example, and become deeper computations—abstractions, mathematical formulae, and so on, as collaboration progresses. The first, gentle computing artefacts are scattered, flexible and ephemeral, and as computation gets deeper the artefacts are linked, coherent, and more structured.

Returning to the sketch, at the outset of the collaboration, there is little overlap between these conceptual spaces—a "gulf of interpretation" as [IT4] put it. However, as the attuning process takes place, either the artist will gain sufficient technological understanding to tell the technologist what he or she wants or (more usually) vice versa—the technologist gains enough understanding of the artist to figure out to make what he or she wants. Once shared understanding increases (it is not necessarily mutual, however), for the technologist to know (or be told sufficiently clearly) what to do, he is able to make a gentle computational artefact, with gentle artistic meaning.

This artistically and technologically gentle computing artefact lives in the shared conceptual spaces of the technologist and the artist, and encapsulates small, particular elements of each person's respective deeper understandings. Starting from this anchor, or boundary object, it then becomes possible for the technologist to navigate further into the artist's conceptual space, creating further boundary objects with deeper artistic and/or technological meaning, and potentially the artist can do the converse.



In larger collaborations there are more people's conceptual spaces in play, and more opportunities to weigh anchors of meaning. One such example, involving an HCI researcher's perspective, is perhaps alluded to by [IA1]:

> *[it] has been great working on [Art title] with [HCI researcher] and her mob [Her perspective is] of course to think that there is a much broader idea of interaction design […] and then the GUI will take care of itself because it's a micro of that sort of thing.*
>
> [IA1]

Aside from the artist gaining an increasingly deep "artistic" relationship with the artefacts, and the technologist gaining an increasingly deep "technological" relationship, there is also the possibility for each to gain deeper relationships in each other's fields of expertise.

> *I try and bring as much as I can aesthetically and creatively to a project as well as try and keep the technical side of things as permeable as possible as well*
>
> [IT4].

The extent to which this happens or is desired is varied. Technologists are more likely to engage artistically than vice versa (at least in part since the artist is generally higher in the collaboration hierarchy), but each direction has clear trends in the data.

ARTISTS EXPLORING TECHNOLOGICAL STRUCTURES: NAÏVE INTERACTIVE ART, AND HUMAN COMPUTATIONAL INTERFACES

There was some discussion in the interviews about whether artists should learn to program or not, with opinions varying (for my own discussion, see pp. 69–72). Generally artists were characterised as either seeking understanding of the 'back-end', of computation, or being interested in the 'front-end' audience experience only. Most strongly in support of artists' back end learning was [IT3]:

> *by better understanding what I'm doing [artists] can better design their own works and better understand the constraints on them […] artist practice is always better from them engaging in the[ir] technologies … the more they learn, the better art practitioners they'll be […] it becomes on the heads of people like myself and yourself to help [artists] get educated in the same way they help educate us in the making of art.*
>
> [IT3]

Though [IT3] acknowledges that reality doesn't match his ideals, and that artists tend to be much more interested in the front end of the computing system:

> *80% of the time the artist wants to be involved in everything […] not programming [but] how the decisions I'm making are going to affect the outcome"*
>
> [IT3]

[IA1] also concentrates her interest on the 'front end', though she expresses frustration at meeting that boundary:

> *the hardest thing [about working with programmers] is that there is a point at which I can't help with the problem or engage with the problems", and later, "is this going to impact on the front end, the functionality of the interaction, or is this a back end thing? And like no it's a back end thing … sometimes I get frustrated with my own lack of knowledge because I can't help with that sort of thing*
>
> [IA1]

Overall, she has learnt a great deal generally about computing, but acknowledges her own conceptual constraints, and the radically different conceptual spaces of computer scientists, as the following three quotes illustrate:

> *I've learnt generally a lot about computational systems and generative systems [etc].*

then,

> *I've learnt a hell of a lot from programmers but I'm not looking at them to teach me code…*

and,

> *I wonder if [artists who want to program are] really respecting that that person has done 4 years of comp sci […] it's a whole way of thinking, and a whole way of conceptualising*
>
> [IA1]

The artists I interviewed were well aware of the impact that computing has had upon the arts:

> *there are specificities about computational processes… which have deeply impacted on, particularly time-based art but also 2D arts as well […] computational process have made us rethink linearity*
>
> [IA1]

and,

> *Once I understood how the computer could be used for this systems dynamics sort of work I fell out of love with filmmaking*
>
> [IA2]

Yet the artists' verbalised awarenesses of computing did not seem to extend far beyond simple, obvious differences between computing and their 'prior' art (film-making). [IA1] characterised computing as making artists 'rethink linearity' and as a labour-saving device (see "Servitude", one of my four Ss (see pp. 146–149)), and [IA2] was attracted to computing as "show control".

> *Overall there was a strong current of feeling that there was some benefit in artists retaining a degree of naïvety about computing:"my naïvety actually sometimes might be beneficial because one gets into habits of saying I know what's possible, I know what's not possible*
>
> [IA2]

In the traditional art world, naïve art has apparently gained a measure acceptability, though it is known for its awkward and often refreshing vision[5] . To a technologist, an naïve artist's vision may be similarly awkward yet refreshing, but perhaps some of the advantage of collaborating with an expert and professional technologist is that he is able to counter the negative aspects of such naïvety. On the other (other) hand, with painting the advanced potentials of the medium are physical, allowing naïve (untrained) painters to discover and explore these potentials. Yet the advanced potentials of computing are abstract and structural, and could be less likely to be discovered or explored by naïve artists.

After all this conflict, if the artist still wishes (to learn more deeply about programming), he or she can use the same physical anchors to navigate further into the technologist's conceptual space. There is evidence of this happening from the interviews:

> *I'm happy always to [answer artists' questions] but I often stop short of going any deeper and I use kind of metaphors and I use kind of disparaging language to keep them out of the nitty gritty that would take too long or too much energy to explain […] sometimes they'll push further and push further so you kind of give them incrementally more information till they're satisfied but you always start with the*

5 http://www.daprix.com/barton/artist/naive.html

> *smallest amount of information to give them the idea and to save you time.*
>
> [IT3]

Recalling my first study, which argued for the need for computing to have the ability for a user to navigate a continuum from gentle to deep programming, it is tempting to remark that [IT3]'s approach provides precisely this ability—to expose depths of computing only to the extent that artists' conceptual spaces require it. However, [IT3]'s actual ability to explicate every aspect of computing in this way is unknown. [IA1]'s quote, above, about learning 'a hell of a lot' from programmers, but not wanting to learn code shows that a similar approach may have worked in her case.

A comparable approach is that of [IT2]:

> *my experience is more to do with artists that have a high-level idea for the behaviour of a system, and an application that they would like to have to create their work (often media of some form). So for me, the design interaction with the artist is more a matter of conversing in terms of the artist's experience of other software applications"*
>
> [IT2]

There is a possibility that explaining the designs of systems in such a way, i.e. in terms of other media creation software, would perpetuate the kind of naïvety that restricts artists' awarenesses of computers in the same ways that the interviewed artists' awarenesses seemed to be (see earlier in this section). This may not be desirable if the artist is curious to be taught about computing.

An interesting point was made by [IT4], in response to a question I asked about conveying the structure of computing to an artist:

> *realtime response is more about control than program*
>
> [IT4].

In other words, in the thick of generating an output in response to an analysed input ([IA2]'s simple 'show control'), there is possibly too little

time or space for an artist to detach his or her conception of computing from the present and physical in order to consider the 'grand beauty' of computing's abstract structural properties. The added physicality of real-time visual programming languages is unlikely to help this state of affairs.

Another simple technique that technologists use to expose technological structures to the artist is to offer examples or demonstrations of software functionality. In my interviews, [IT3] described using his (not always first-hand) knowledge of interactive art production to suggest useful approaches to artist. [IT1] showed an artist Max/MSP help files (which are fully functioning, interactive Max/MSP programs) in order to demonstrate the capabilities of that software, with an unintended consequence:

> *[The artist was] saying "I'm wondering what you could get out of this video that we're analysing" […] and I'm saying "well, we could do anything. Anything you can think of". And of course it's very difficult to think of anything then, so I've been going through the examples in Max which every so often we come across one that he really likes. And so I'm showing him those to give him an idea of the kinds of… that you really can do anything […] but then there's always a problem with that because I've shown him the demo, it's just the help file […] and the he keeps saying "This is great, I must go and show people", and then I feel like a fraud, you know, because I haven't done anything, I've just showed him.*
>
> [IT1]

This is a return to the naïve art issue—the artist is seeing for the first time what would have been quite familiar, not to say clichéd, to people experienced with Max, and, according to [IT1] has let himself become inspired by the novelty. There is a possibility that the technologist may have not only demonstrated the possibility to do 'anything you can think of' with video, but at the same time profoundly biased the artist towards one or more of the video processing demonstrations.

TECHNOLOGISTS EXPLORING ARTISTIC STRUCTURES: INTIMATE ITERATION AND COMPUTATIONAL TOY-MAKING

Having established the initial shared technological anchors, creative interdisciplinary collaborators proceed using combinations of two different techniques, which I call 'intimate iteration' and 'computational toy-making'. (I am leaving aside the media production and traditional iteration (engineering) methodologies, which were characterised earlier as being less satisfactory, more appropriate for straightforward development phases, and less supportive of creativity). Intimate iteration is fairly similar to other agile programming methodologies, and is an extension of the to-and-fro attuning technique used to establish the initial anchors. The program development proceeds in many short cycles, with the artist and technologist working in close proximity ("in situ" [IT4]), and successively approximating the final piece by creating many small anchors of shared meaning. This shared meaning emerges at increasingly deep levels, as individual deep meaning is also continually revised, as [IA2] describes:

> *In a kind of programmatic way or a named way I continue to be attracted to the philosophies and some of the techniques of agile programming, principally because I see it as a means of operating whereby you do have a holistic understanding of the eventual delivery but you address immediate and evolving needs rather than fantasising that you can describe a project in the brief right at the start. So that slippery sort of heuristic finding of the project at the same time that you know that you have to envisage the project, I find that really fascinating.*
>
> [IA2]

[IT1] likens the iteration process to sketching:

> *I make things up as I go along, usually, I think. But I think that's more my designery [sic] training because with that you have a vague idea, then you, like, draw it and then look at that and discover something new in the drawing.*
>
> [IT1]

[IT4] "exposes the important parts of [the] development work to the artist as early as possible" (i.e. not after a lengthy development cycle) in order to resolve ambiguity and difficult design decisions.

During the course of intimate iteration, the process of attuning between artist and technologist continues. As the technologist becomes more familiar with the artist's conceptual space, he is likely to feel more confidence in making autonomous suggestions or changes that affect the artwork. Presumably most suggestions or changes are respectful and appropriate to the art, but the evidence throughout the study is that technologists can be motivated by technical, as well as "front-end" factors—which is which is not always apparent to artists (nor, for that matter, to the technologists who are acting, except in hindsight). In one sense a technical motivation could also be aesthetic—if a given technique is more computationally efficient or elegant, a technologist may prefer it to one that produces nicer-looking graphics, for example. On the other hand, technologists, particularly in resource-compromised situations, report occasions when they have made suggestions or changes in order to do less work.

However, when they responded, artists seem not to be swayed by suggestions or changes that are technically motivated if the "front end" is compromised ([IT1]: "he didn't let me get away with it"). Yet this also depends on an artist having an adequate enough perception of the capabilities of computing to be able to distinguish technologist unwillingness from technological achievability.

In the COSTART reports, there were instances of artists being frustrated by a lack of a 'hands-on' approach to art-making, when watching a technologist work during intimate iteration. Computational toy-making encourages a far more hands-on approach from the artist..

The concept of computational toy-making emerged from the responses to my questions about whether the technologists added things to the code to make it easier for the artists to engage with the software, combined with a specific example of this I saw in the interviewed artists' project. The technologists' answer to my questions were universally 'yes',

they made parts of the program for the artist to engage with, though interpretations varied. [IT1] expressed a wish to "hide code" (and leave a simplified version for the artist). [IT2]'s most recent collaboration he described as a "content creation" application—the entire program was for the artist to use (this is sometimes called meta-interactive art). [IT3] expressed a goal to make the program understandable (self-explaining and transparent), and in his current art collaboration [IT4] created the specific example which inspired me—a graphical interface for manipulating parameters of an algorithm.

Similarly, in the responses to the question "do you show artists the code?" the only examples where this had happened were when the code was in a visual programming language such as Max/MSP or PD. As [IT4] describes it:

> *[Visual Programming is] a domain where artists feel confident enough or have an interest in becoming conversant", and later, "it's possible to use the environments and understand the visual syntax without really having a good sense of programming or structure*
>
> [IT4]

Visual Programming Languages (VPLs) provoke understanding and familiarity in artists in ways that textual code does not. In terms of the parts of the program's graphical appearance, the buttons, sliders and mouse interaction are familiar from everyday user interfaces (with connotations of direct manipulation and undoability, et cetera encouraging 'risk-free' exploration). Structurally, the program is rendered as a chart, which (in simple cases (Shneiderman et al., 1977)) is more readable than textual code, and (even in complex cases) is reminiscent of non-computational charts, where paths can be traced with a finger. As [IA1] pointed out, they remove the conventional need for separate diagrammatic documentation. Even if a reader of the code is unfamiliar with charts, there is a strong physicality—resonances with pipework, road maps, machines, and so on. All these things help to inspire confidence.

[IT4] used an interesting word: "conversant". It evokes the notion of conversation with a programming environment—in other words, that the program environment responds in a realtime basis, which is what

Max/MSP does better than many other languages (because it removes aspects of the compiler wall—see Study 1, Chapter 5). Adapting to the mutual feedback provided during the conversation is yet another form of attuning, this time between technologist (who may be an artist) and program.

[IT4] created the interface for manipulating the parameters of an algorithm in response to his observation that "no one involved in the project had a good understanding of the sort of algorithm that had been chosen and discussed"—in other words there was a lack of shared meaning—a gulf of interpretation to be crossed, that appeared not to be crossable by attuning between people. So [IT4] decided to take himself out of that attuning loop, and to give control of the algorithm to the artists by way of a control panel. The specific aim:

> *critically… it makes it really clear what the dynamics of the system are as opposed to what my interpretation is as well*
>
> [IT4]

There was an additional, possibly somewhat passive-aggressive motivation for [IT4] to take himself out of the loop:

> *this was a case where the choice of the algorithm and some other technical aspects of the work were specified in the process of designing the work, which I don't think is necessarily appropriate particularly when there wasn't a great deal of knowledge or experience with these technical artefacts like algorithms to begin with. It was just a case of saying, on the one hand, "Here's what you asked for, here it is," and making it clear that other things could be done but this is the first part, so doing what had been asked for, rather than getting trapped in the artist wondering whether what they were seeing was just the limitations of my interpretation or of the algorithm or other things.*
>
> [IT4]

So by taking himself out of the attuning loop, [IT4] could justifiably claim, if necessary, that the unsatisfactory result was the effect of an inappropriate design choice, not of implementation or interpretation.

When I asked the artists about [IT4]'s interface, both their responses involved notions of 'play'. This aligned well with the COSTART artists', and interviewed artists' earlier reports of playing with computing media (especially Max/MSP interfaces) to analyse problem spaces and "develop rules" ([IA1]) in an immediate and exploratory manner. Incidentally, "Developing rules" is not always an explicit aim in interactive art collaborations. [IA2] describes wanting to work with "dynamic sets of information that had probabilities and tendencies in them rather than prescribed discourses" [IA2]. Fortunately, [IA2] indicates that the same interfaces help this exploration through attuning, too:

> *often the artworks that are being made are manipulable systems of some kind. I keep coming back to this metaphor, they're things that need to be tuned in particular ways and often the tuning is best achieved from my end of things if I've got a kind of array of sliders by which I can adjust things.*
>
> [IA2]

[IT4] adds that "a lot of the character of the system actually comes from that sort of control" for the artist. As a result of the playful connotations I decided to call these technologist-constructed interfaces for artists "computational toys". Although [IT4]'s computational toy was for a single algorithm, [IT2] constructed an entire application designed to be a 'content creation' system for an artist. I would also characterise this application as a computational toy. I do not wish to demean the usefulness or importance of these interfaces by calling them toys, however. On the contrary, as Resnick et al. relate (albeit in a more general context),

> *simpler (and better designed) tools are often denigrated as "toys"; the mouse was widely derided as such, at the time of its introduction [Resnick et al. provide no source for this anecdote.]*
>
> (L. B. Resnick, Levine, & Teasley, 1991, p. 29)

So my conception of computational toys is neither 'dumbed-down' nor 'useless'.

The artists' response to computational toys was positive. [IA2] finds that he has "a better understanding of what the algorithm can make hap-

pen", indicating that the technologist is successfully out of the loop, and that the potentials of the algorithm were made more clear.

Crucially, as [IA1] stated,

> *instead of [IT4] just doing it and you saying 'can it be more squiggly?' and him going back and changing parameters, I found I* understood the language of the algorithm *just by playing with the parameters and understanding what the software developer, how they had broken down this organic thing*
>
> [IA1], my emphasis

The artist was not referring to the language in which the algorithm was implemented (Python, incidentally), but rather the language necessary to communicate meaning to and from the algorithm itself. This attuned manipulation of an algorithm's own language produces computational meaning between the artist (and other collaborators using the toy) and technologist. [IA2] phrased it thus:

> *everyone who has responsibilities and curiosities about the project has some way to gauge impact and requirement […] you can mess with it and move the sliders around and see how the world's altering from [your] point of view […] those settings are for everyone. Really that's a way to gather around something concrete*
>
> [IA2]

Toys are shared objects. As such, they don't encapsulate every nuance of everyone's meanings, but configurations can be interpreted by individuals in terms of their own structures of meaning.

Summary

Broadly speaking, the grounded theory contends that of fundamental importance is the process of attuning between humans, and between humans and technology. In particular the technologist must ideally reach a holistic understanding of the art system across many continua: from artistic concept to technological implementation; from front end to back end; from people to technology; from gentle programming to deep programming. The technologist bears the brunt of the responsibil-

ity for attuning the sides of these continua, and, poorly managed, this can easily result in frustration or powerlessness on the part of the non-programming artist. Technological development of art systems starts with a metaphorical description from the artist, which is embodied in successively deeper computing systems by an attuned technologist. The technologist also attunes the computer to the artist, through a combination of 'intimate iteration' and (preferably, for creativity support purposes) 'computational toy-making'. Ideally the attuning happens to the extent that the artist no longer needs the technologist, yet this does not always happen in practice because of the extra work involved in making toys and the lack of specification of them.

'Playing' with technological 'toys' is crucial to the development of interactive art systems, for these reasons, most of which are particular forms of attuning:

1. finding the rules that govern the behaviour of the algorithm,
2. developing the 'tendencies' or character of the system,
3. intuitively learning the 'language' and capabilities of the algorithm by:
    a. exposing the potentials of the algorithm's dynamics,
    b. making the algorithm's place within the system apparent—highlighting the structural interactions and limitations
    c. providing a realtime response to artists' actions
4. producing computational, technological and artistic meaning simultaneously (which reduces ambiguity, produces successful boundary objects, and so assists interdisciplinary attuning),
5. making the artist feel more comfortable and empowered,
6. incorporating synthetical and analytical approaches to problem solving, and
7. taking the technologist's interpretation out of the loop.

These toys, then, are boundary objects that embody a language for creating meaning between artist and technologist, and artist and computer, and perhaps less necessarily, technologist and computer. (There is also scope for these toys to become boundary objects for other communities, such as audience members, other artists and technologists, researchers and curators).

In conclusion, the theory has given us a new way to understand artist–technologist collaborations as dynamic processes of attuning through social relations, the technologist's role, and through technology. It remains for us to find ways to exploit this new understanding in the context of technology that supports computing as a creative medium, and the next chapter is a first step on that path.

8. Design Recommendations

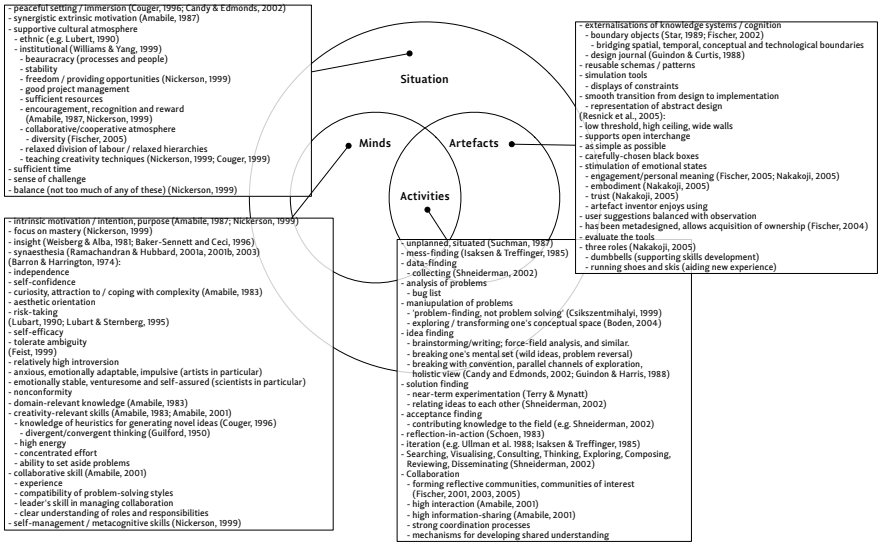
Introduction and General Comments

The first study, Chapter 5, presents an argument for rethinking the mythologies surrounding computing technology in general, and compiled programs in particular, in the light of the need to support creative work. The second study, Chapters 6 and 7, relates the experiences of technologists and artists in collaboration, with the aim of identifying particular ways to support these collaboration through technology and methodology. Together, these technological and social understandings of interactive art development provide a holistic insight into the problem of designing creativity support tools. In this chapter, I draw on that holistic understanding to capture recommendations for supportive methodology and technology for creating interactive art.

Before I do so, there are a number of factors to bear in mind. The first is the chicken-egg question of "which takes precedence? Methodology or technology?"—in other words, in interactive art, should we make technology to support art-making methodology, or make methodology to support interactive art's engagement technology?

I believe that both processes need to happen in interactive art, but in a particular order. This stems from the idea that, as we found in Chapter 5, interactive art (and other creative computation) most often engages with the computer for reasons that are integral to computing (for example, the 4 Ss: speed, servitude, synaesthesia, and structure)—no other medium will suffice. To 'redesign' this medium makes little sense at this stage, since a) computing is flexible enough to support a wide range of creativity b) the computing medium is still full of untapped potential for such creativity support and c) it is that potential that is often engaged with in interactive art. So when I talk about designing technology and methodology, in the first place I mean *re*designing the technologies that prevent engagement with the computing medium. In the second place I mean capturing requirements for methodologies that encourage engagement with the computing medium. In the third place I mean designing technologies that are aimed to support such methodologies, in the sense of automating and augmenting common activities.

Let us return to the Creative Environment Diagram developed in the first two chapters (a more legible version appears at the very end of Chapter 2).



The diagram shows features of situations, minds, artefacts and the activities that happen between minds and artefacts that are thought, from the literature, to support creativity. We can see, in the light of Chapter 5, that the diagram does not particularly capture features of the creative me-

dium itself (in this case computing), except perhaps obliquely as an 'artefact'. On the one hand, this is fine, since the features in the diagram relate to 'creativity support', rather than the 'object' or 'muse' of creativity; on the other hand, part of what is so special about the computing medium is that it has integral qualities of creativity support. So in these design recommendations, the first task is to bring about the possibility to engage with these creativity support qualities of computing, and this is likely to be most effectively aided by implementing a system which aligns with the diagram's features of situations, artefacts and activities that support creativity.

Another factor to bear in mind is the prior statement that for this research we are not interested in designing new situations in which to be creative, nor are we interested in designing ways to make minds 'more creative'. The reasons these features are included in the diagram is because creativity support features of these parts of an environment should also have a bearing on parts that are contained within them. Thus, the qualities of supportive situations also become qualities of supportive artefacts (or technology); and the qualities of supportive artefacts and supportive minds become qualities of supportive activities (or methodology).

A third factor to consider is Brock Craft's warning of the dangers of blindly following guidelines and assuming that doing so automatically makes for an appropriate design. To that end I will be evaluating my designs in the context of a new art collaborations, described in Chapter 9.

Should we help artists to learn to program, or should we help programmers capture artistic meaning in computational terms? These two approaches needn't be mutually exclusive. The aim of both is to give the artist complete and expressive control of the computer. At the moment both are markedly insufficient, requiring immense and often aesthetically irrelevant technical knowledge. An analogy with painting is that if the painter requires to mix his or her own paints as an important part of the artwork, then he or she will do so, otherwise asking an assistant to mix, or using oil paint in tubes may suffice. Deciding whether or not to learn programming, or to mix paints, is an inherent part of constraint-

changing creativity, and the potential always exists to want to change the constraints. I will therefore adopt the position that emerged from the grounded theory study, and which is corroborated in the literature (see pp 69–72), that artists needn't learn to program if computing *per se* isn't a key conceptual part of the work (and presumably, for non-programming artists, it isn't). It is important in creativity support design not to deny the ability to go "back to basics" and mix one's own computational paint.

There are some overarching design characteristics of interactive art, garnered from the literature review, that tools to support it should allow. These are:

- Realtime response, to inspire interactive engagement, and to make clear the relationships between input and output.
- Robust (the artwork shouldn't crash and thus require manual intervention to get working again).
- Communication between a variety of systems (unusual and high-performance input/output devices, networked information and static data)

Successful implementation of these characteristics is dependent on the skills of the technologist, but it makes little sense for them to use a computing medium which is inherently too slow to give realtime results, too unreliable not to crash, or too closed to prevent communication between a variety of systems.

A Computing Medium for Creative Engagement

As Chapter 5 argues, the computing medium has the potential to greatly support creativity, particularly because of the speed, servitude, synaesthesia and structure it has as a unique combination of qualities. These qualities give computing flexible constraints, which it is useful to have in creative tools. Yet a conventional mythology of computing puts a compiler wall in the way of these flexible constraints, meaning that users are unable to easily transition into programming and are thus creatively limited. The compiler wall imposes *conceptual* and *physical* (*temporal* and *spatial*) separations between programming and using, none of which are

fundamental to computing, and we need to mitigate against all of these aspects.

Chapter 5 further claimed that Smalltalk and Max/MSP between them show that each aspect of the compiler's separation between programming computers and using them is not fundamental to computing.

The grounded theory study documented in Chapters 6 and 7 also contains instances of technologists desiring that the compiler wall should be abolished, and that end-user programming should become more accessible. [IT4] wishes to have the 'the best of both worlds' in programming languages:

> *there are certainly gaps and gulfs between the different programming development environments and paradigms and techniques that I feel very keenly. Lots of things that I enjoy about patching environments […] I would like to have, to some extent, in C++*
>
> [IT4]

When asked if he there was anything missing from tools to support artists engaging with programming, [IT3] answered:

> *there's a myriad of things missing from tools, the main problem is that […] there's quite a singular decision about how you go about solving a problem, […] there's always the best solution for each person to each problem […] only I will ever understand how [I solved a problem] because I come to it with all of the baggage of the things that I'm good at, the things that I'm familiar with, code I've already got written to do, certain library, certain toolbox functions, that sort of stuff.*
>
> *So having a really simple but incredibly wide and powerful toolbox to do all sorts of things, and to have everyone using that same beautiful mythical toolbox to do things, would make transitioning between things, between other people's code and mine, […] would make it really easy. But things like that don't exist, because everyone's using different programs and different ideas and [has]*

> *different backgrounds, and even in different physical and spoken languages like you often end up doing things that other that a hundred other people have already done before, and that could and should be avoided. How it is that you go about avoiding it?*
>
> [IT3]

Earlier, [IT3] had said,

> *the hardest thing about programming […] is curbing my urge to just start straight away […] it's dealing with the operating system or […] protocols or […] vagaries. To get the shell right takes much more time than you expect.*
>
> [IT3]

[IT3]'s wish for a simple 'mythical' toolbox that everyone uses, to help him avoid redoing things that 'a hundred people have already done before' comes as a result of the 'tower of Babel' syndrome of modern programming languages and applications. Despite the fact that all programs on a given architecture use the same processor instructions and memory, the inability to easily 'lift the hood' on complex languages and applications means that they become opaque and fixed, rather than transparent and reusable. It is easier to recreate something than it is to reuse it. This is why [IT3] takes more time than he expects on the 'boring' parts of setting up the shell. Even programmers need assistance in understanding the existing, less-interesting parts of computer systems!

As far as [IT3]'s "simplicity" goes, it seems sensible for simplicity's sake to adopt a single, universal paradigm for creativity support programming. By universal, I mean that a paradigm can be used for programming at the very atomic level (processor commands), at the human level (interface manipulation) and at the abstract level (recursion, data structures, and so on). No paradigm fits the 'universal' criteria better than object orientation. Recalling Alan Kay's comment:

> *… each Smalltalk object is a recursion of the entire possibilities of the computer. Thus its semantics are a bit like having thousands and thousands of computers all hooked together by a very fast network … For the first time I thought of the whole as the entire computer and*

> *wondered why anyone would want to divide it up into weaker things called data structures and procedures. Why not divide it up into little computers, as time-sharing was starting to?*
>
> (Kay, 1993, pp. 70–71)

In my grounded theory study, two streams of thought emerged on the use of object orientation in interactive art development. One view is that the paradigm (or any sort of abstraction) isn't crucial for a lot of interactive art, because,

> *once you're working in flat system-level patches with real-time response, it is more about control than program. You're not having to plan and make delayed decisions about building things up into final forms and ensuring they operate when everything is working together*
>
> [IT4]

In Max, the concept of object-orientation is partly built in, in terms of sending messages between autonomous processes, but the ability to reason more abstractly about objects is not provided. Another view is that object orientation is extremely useful and should be used wherever possible.

> *[a change to one class] was the only simple change I needed to make instead of having to change every single [object]. And so that kind of turned me from not quite understanding what object oriented programming meant to wanting to only use that, because then I knew, it's definitely the way to go.*
>
> [IT3]

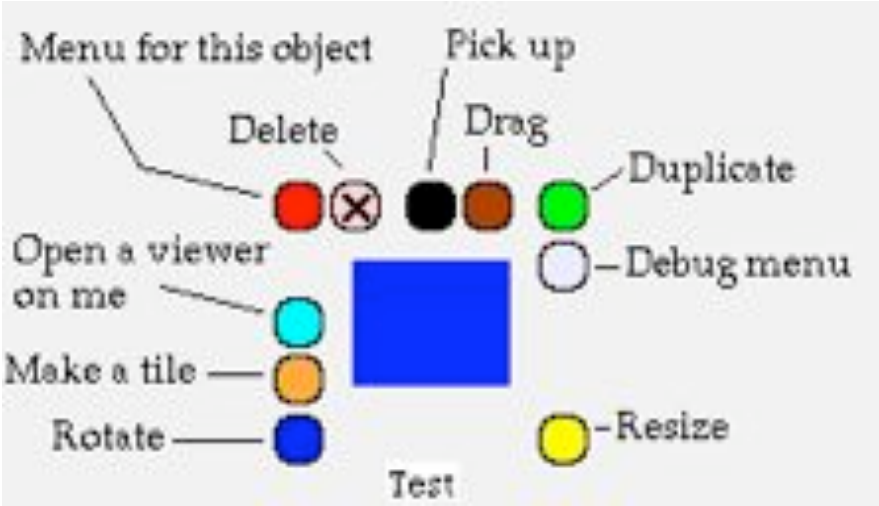
[IT3] had said earlier:

> *So what would be beautiful would be if I could write software that was so transparent that the people using it could see inside it without my help. That would be a great goal*
>
> [IT3]

Being able to "see inside" software implies many things: most obviously open source code, and documented code, but also an interactive design in which the code is separated as little as possible from its computation-

ally running result. Once the code is visible and understandable in this way, it is a short step to making it reusable.

So how can this transparency be achieved? On a completely object-oriented system, one way is to implement a "lift the hood" command, available on every input or output object, which shows an interface for exploring or editing that input and output object, plus the objects that comprise it and the objects it is connected to. Smalltalk already has this feature, in the form of the 'halo' .



A schematic of Squeak Smalltalk's 'Halo'. The blue rectangle is the object to which the halo controls pertain.
(image retrieved from http://static.squeak.org/tutorials/morphic-tutorial-1.html)

The halo can be used for graphical operations—to manipulate the size and shape of an object, for example, but also to bring up Debuggers and Viewers, for exploring and changing the Smalltalk code describing the behaviour of an object. The halo surrounds the object, and the editors it spawns are spatially close, removing the spatial separation between programming and using each object. Additionally, as in Squeak Smalltalk, the interface for exploring and editing the object can *itself* be explored and edited, opening a gateway to changing the language itself.

Object-orientation, combined with interactive code explorers for each object, further implies that each object's code can be compiled or interpreted individually, being, as Kay puts it, "a little computer". This means that compilation time is reduced since only a single object needs to be compiled, and that during compilation, other objects in the system may

keep running as normal. This would remove the temporal separation between programming and using each object.

So Squeak Smalltalk's architecture is fairly close to a computing medium suitable for creative engagement. However it still imposes a subtle separation between programming and using, in that whilst objects in the graphical user interface are graphical, the Smalltalk language that describes their behaviour is textual. To maximise consistency, the language for describing objects depicted on screen should be graphical, yet this is at the expense of the advantages of textual languages, namely speed of manipulation, detached reasoning, and consistency of layout. I propose, therefore, that as Balaban et al's treatment of intentional and extensional editors suggests (see p. 73), programming languages should present *both* textual and graphical interfaces, according to the programmer's preference at any time. Programming language and programming interface designers should thus consider how to map and to manage the transitions between these modes.

Methodologies to Collaboratively Engage with Computing

From my grounded theory study, it emerged that creative collaboration rarely uses a single style of co-working. Mamykina and Candy's categories of equal partnership, equal partnership with artist control, or programmer as assistant all appeared at times, but other relationships included programmer as teacher, programmer as project leader, separate work, and so on. The relationship between collaborators was dynamic and contingent upon different stages and circumstances of development. Therefore methodologies for making interactive art should support these dynamic power shifts.

From the literature review, we discovered that creative people have relatively high introversion and independence—this sounds like bad news for collaboration! On the other hand, John-Steiner (John-Steiner, 2000), says that in creative collaboration new ideas are generated through a shared struggle.

Evidence of this conflict emerged in the grounded theory study too. Some artists occasionally felt frustrated by not being 'hands-on' with the computing (i.e. by not being independent); some technologists were

'annoyed' by pressure from the artist (indicating a need to work alone). Yet most were motivated to collaborate by the challenge and mutual learning opportunities.

Clearly methodologies for creative collaboration need to provide scope and space for both individual and shared working. However In the situation of creating interactive art, current technology biases this scope: the artefact that is being created is computational which, conventionally, gives the technologist much more ability to work alone than it gives the artist. Therefore methodologies for doing this sort of work need to restore the balance of power between artist and technologist, by promoting techniques to allow the artist to engage with the computational artefacts, without needing a technologist to constantly mediate.

Fischer's metadesign (Fischer, 2005a; Fischer et al., 2004) ("designing for designers") provides a general framework for designing systems that creative users can modify and evolve, allowing participants to 'transcend beyond the information given to incrementally acquire *ownership* in problems', and to 'shift in focus from finished products or complete solutions to conditions, contexts and tools'. In my study of interactive art collaborations, there is a great deal of evidence in favour of this shift in focus, and we can use this evidence to make specific recommendations for approaching interactive art collaborations from a metadesign perspective.

In summary, I recommend that metadesign methodologies that involve toy-making particularly are recommended. Metadesigns are developed in a moderate iterative cycle (which I call 'traditional iteration'). In collaborations I studied where no metadesign was happening, the development cycle tended to be more like one of the agile development methodologies, what I call 'intimate iteration'. My contention is that in intimate iteration, the technologist becomes too caught up in making the artwork to be able to make tools for the artist. This is not always a bad thing, and is often apparently desired, but in the case of the toy-making metadesign that I recommend, it is, because it denies the artist opportunities to become more directly engaged with computing or with the artwork's manifestation if he or she wishes. Rather, I recom-

mend that artists consider collaborating with technologists to create tools for making art, instead of making art directly.

Firstly, programmers in the collaborations are very motivated by completing work—achieving goals and solving problems, and were attracted to having control over 'hermetic' systems. As [IA2] puts it:"part of being a good programmer is often the ability to go into a somewhat hermetic world and enjoy setting up that world and finding focus in it" [IA2]Yet from the artists' points of view, there appeared to be much less of a goal/solution orientation to the creative or artistic problem. This resulted in situations where programmers were frustrated by not having a sense of completion:

> *I did enjoy the [Artist name] one, but it was annoying that we hadn't got it finished. He kept standing there saying, "it's such a shame that we haven't got it finished", while I'm frantically trying to do stuff.*
>
> [IT1]

> *if someone sets me out a goal that I find hard to nail down … [it's] going to make me unsatisfied*
>
> [IT3]

> *I would have as least favourite [art projects] are probably the things that I've spent a lot of time on and haven't actually managed to find a good form to move away from them or to feel a sense of completion or for them to actually go through that process of translation from internal to external.*
>
> [IT4]

This frustration appears to stem from the artist expressing problems in imprecise terms, leaving the technologist feeling out of control, with little idea of how to solve the problem, or how long it might take, or how much work a solution might involve.

There is a tendency for artists towards solving problems in interactive art collaborations by 'analysis'—by intuitively exploring what already exists. Technologists, on the other hand tend towards 'synthesis'—a con-

sidered bringing of new things into existence. As a COSTART artist put it:

> *There are authors who prefer an approach that is completely free but also quite arbitrary, and others who on the contrary prefer to follow precise rules. As always only the results are important, as they tell us if a certain approach actually works. I believe that a way in the middle is the best solution.*

The tendencies are not universal, and could easily be explained by the technologist's relative expertise in synthesis with computing, compared with the non-programming artist's lack thereof. Yet in all collaborations there were occurrences of both, leading me to believe that balancing an 'arbitrary' approach with a 'precise' approach *must* involve iterations of each. It follows that methodologies for making interactive art should mingle both approaches, to avoid either party's frustration at 'only making things' or 'only fiddling with things'.

A way to overcome such frustration is to have the technologist *meta-design* a system for the artist. In other words, rather than directly helping the artist to make the artwork, the technologist should, as much as possible, 'synthesise' an environment for the artist to actively 'analyse' to make the artwork. The idea behind this is that it would be easier for the artist to describe the tools he or she needs to be able to make the artwork, than it would be for him or her to describe the artwork directly. This approach is supported by evidence from my study of collaborators, most clearly in [IT2]'s account, and [IT4], [IA1] and [IA2]'s collaboration:

To take the latter first, [IT4] aims to "get a good sense of which parts [the artists are] interested in, making [them] extendable or expandable". One manifestation of this approach was the interface for adjusting parameters of an algorithm, which inspired my conception of 'computational toys'.

Methodologies that encourage toy-making are recommended for interactive art development. Computational toys are graphical interfaces, either manually or automatically constructed, for users to 'play' with the parameters of algorithms, or computing sub-media (see Chapter 5, pp.

145–146 for a definition of sub-medium). A toy can be as simple as a collection of buttons or sliders, or as complex as a full-blown content-creation application (which is likely to be the combination of many sub-toys).

Computational toys are boundary objects which produce aesthetic meaning as an artists plays with them, technical meaning as the technologist links the toy to the algorithm, and computational meaning, as the computer processes these manipulations. The result is that computational toys have several possibly crucial advantages over non-playful computational artefacts (this list is developed in Chapter 7):

1. finding the rules that govern the behaviour of the algorithm,
2. developing the 'tendencies' or character of the system,
3. intuitively learning the 'language' and capabilities of the algorithm by:
    a. exposing the potentials of the algorithm's dynamics,
    b. making the algorithm's place within the system apparent—highlighting the structural interactions and limitations
    c. providing a realtime response to artists' actions
4. producing computational, technological and artistic meaning simultaneously (which reduces ambiguity, produces successful boundary objects, and so assists interdisciplinary attuning),
5. making the artist feel more comfortable and empowered,
6. incorporating synthetical and analytical approaches to problem solving, and
7. taking the technologist's interpretation out of the loop.

Providing the artist with computational toys is a metadesign approach. The significant disadvantages with this sort of metadesign are a) that it is badly supported in most programming environments, because they encourage compiled software, and b) that building tools to create artefacts can be harder more time consuming than simply building artefacts (for

example, creating a tool for drawing and colouring squares on a screen is more time consuming than creating a program which draws a particular red square). However, this perceived difficulty doesn't take into account that, in interactive art, the specifications of the desired artefact are often relatively unknown to the technologist, so an artefact may actually take more time to complete, and be less satisfactory, than would a well-specified tool for the artist.

Knowing when to make toys or artefacts is tricky—"knowing how to insert [playfulness] into a structured system is quite hard" [IA1]. If all other things were equal—particularly the amount of time and effort involved—then it would be preferable to make toys because of the advantages listed above. If we adopt a single, universal computing medium that inherently supports creative engagement, as described in the last section and justified in Chapter 5, then metadesign is built in for free. For instance, a program that draws a red square could be deconstructed (by the artist) to find out how it worked. It is not hard to imagine that exploration of the square on screen could reveal a toy for manipulating the algorithm that drew that square—its position, size and colour. Exploration of that toy would reveal the underlying commands that result in a square of any given size and colour. Once discovered, these commands could be altered to produce squares of other positions, sizes and colours. So use of a single universal computing medium, together with an appropriately playful interface, inherently allows any design to be deconstructed into a metadesign which involves computational toys, lifting most of the effort of metadesign and toy-making from a technologist, yet potentially placing the effort of deconstruction onto the artist. Specific recommendations for technology and interfaces to make this balancing process easier will be covered in the next section.

In [IT2]'s collaboration, the collaborators had specifically agreed to create a tool, or environment for exploring 'possible' artworks, rather than a finished artwork.

> *it is difficult to segregate my involvement in the sound that is created from the artist's involvement. To attempt that, I guess I get aesthetic*

> *pleasure from creating the possible behaviours of sound and the tools that are available to the artist to further create the sound.*
>
> [IT2]

This meant that all concerned had a clearer understanding of the requirements at the outset, because rather than being requirements of the artwork, they were *meta-requirements* of the artwork, able to be communicated in far less metaphorical terms.

> *I would say that all involved—artist and programmers did have a fairly good idea of the features we needed before we started. […] the art work was also a content creation application. The application we created was both a tool for creating content and the deployment system that would run the art work autonomously when required.*
>
> [IT2]

Not only would this approach make the artist's specification task easier, and the technologist's problem-solving process more manageable and less frustrating, it would also reduce the technologist's affect on the aesthetics of the piece, and would give the artist tools to work alone more, when desired.

[IT2]'s project progressed along the lines of 'traditional iteration', with the artist even located in a different country. The approach was reasonably successful because everyone involved had agreed to produce a tool, which was easier to specify than an artwork. Generally, the more every collaborator shares an understanding of the requirements, the fewer iterations are needed. However, it is very unlikely that this will be the case for long, since requirements are prone to individual refinement or even to change completely as the development progresses.

So traditional iteration is a poor approach to make an artwork, but is an acceptable approach for making a tool for making artworks. Even then, it is recommended that every iteration involves toy-making, in order to continually test the shared understanding of requirements. If shared understanding is lost, or if a change of requirements takes place, a temporary switch to an agile methodology, such as intimate iteration, is recommended.

Some other projects, particularly in the intensive collaborations in COSTART, were characterised as 'intimate iteration'. Intimate iteration, similar to agile programming (Beck, 1999; Rittenbruch, 2002), is a way for the technologist to 'become' the end-user programming interface, by progressively interpreting the artists' description of the piece in many small iterations. Intimate iteration is a "Wizard of Oz" version of what I call the intuitional approach to End-User Programming (see pp. 74–77), and suffers from the same drawbacks, namely that it relies too much on a non-artist's artistic interpretation, and does not provide scope for the artist to understand or directly control the computing medium. Thus, intimate iteration is acceptable if the artist is comfortable with these drawbacks, but is *not* recommended if the artist would prefer to use a toolkit or to explore the computer to make the art him- or herself. In that case, use of a toy-making methodology, as outlined above, is recommended.

Technologies to Support Creative Collaboration and Engagement with Computing

As I argued in Chapter 5, there are two things in the way of our ability to modify computing sub-media: the ability to understand how to make modifications, and technological 'permission' to do so. The earlier section of this chapter, on designing the computing medium, explains how we can build such technological 'permission' into computing. The previous section contained recommendations for ways to exploit this in interdisciplinary collaborations. This section makes recommendations for technology to support understanding how to make modifications, particularly within this collaborative context.

Generally, we need to be cautious in the design of tools, because they imply particular processes for using them, which may unnecessarily distract from the task at hand. For example, in answer to my question about whether he used traditional software engineering processes in his art development work. [IT4] replied:

> *I use […] compiler and development environments. They obviously imply various things about process in the way they work. I also use*

> *conventional tools like source management and source control tools, which again implies certain processes in the way that they operate.*
>
> [IT4]

[IT4] implies that he is compelled to spend time compiling code, and 'checking in' versions of source code, even though these are peripheral to the process of programming interactive art. Of course, there are benefits to using these tools that make this distraction worthwhile, but there remains the possibility to reduce such distraction. And this possibility is another side-effect of a single, universal computing system, which would allow its own modification. For example, the program that manages the version control could be linked with the program that compiles modifications, meaning that every compilation (or every 10 compilations, or every compilation after a 1-hour threshold) would be checked in as a new version automatically. This is a simple example, but it would be possible to integrate tools to support a task in any way imaginable.

With regard to creative computing, I made the recommendation to incorporate computational toys into development methodologies in the preceding section. With that recommendation comes a need to make the toy-making process as quick as possible, in order to offset its main drawback—that it can take longer to make a toy to explore possible artefacts, than to make a particular artefact.

The technologists I studied had often used visual programming environments to make computational toys for artists, and the artists had been relatively comfortable using these toys. For example, [IT4] says:

> *[Patching is] a domain where artists feel confident enough or have an interest in becoming conversant", and later, "it's possible to use the environments and understand the visual syntax without really having a good sense of programming or structure*
>
> [IT4]

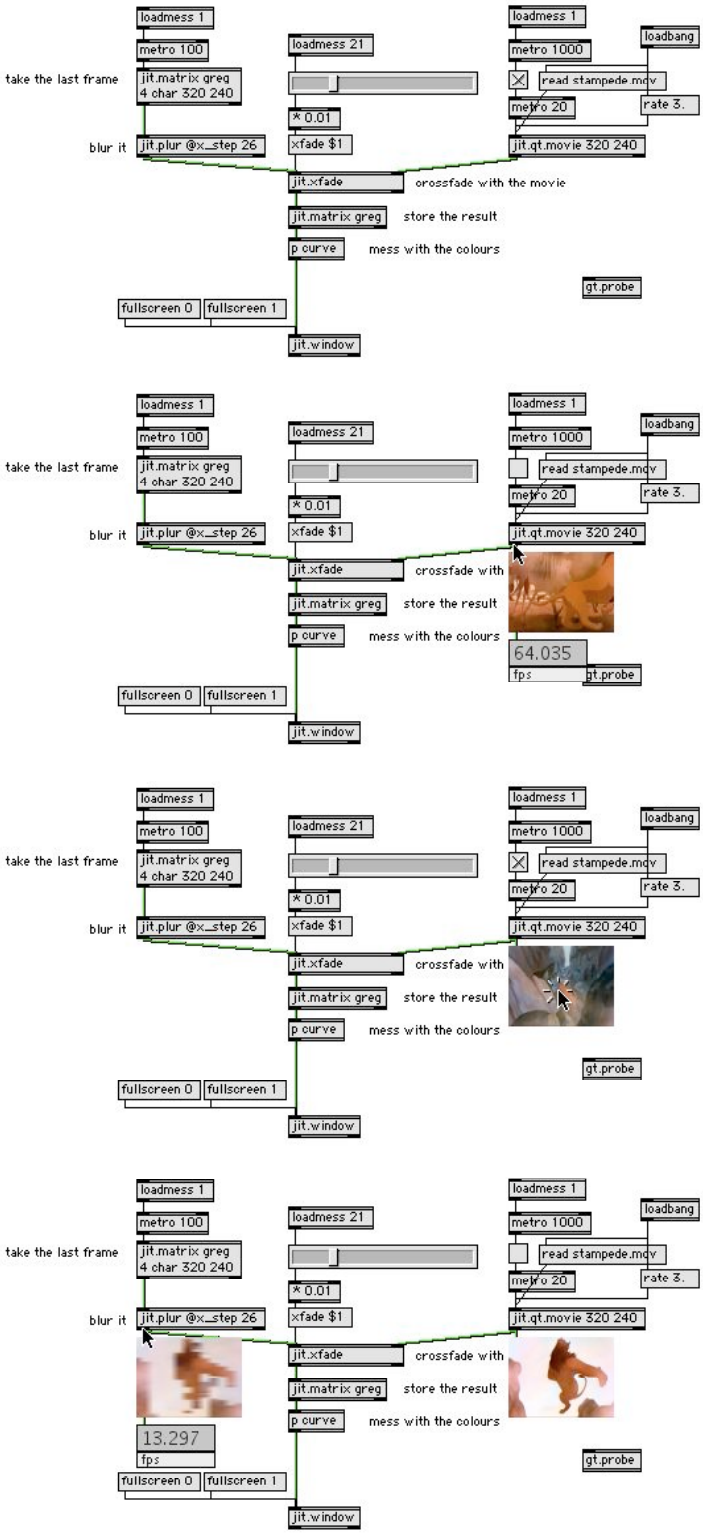
I discussed possible reasons for this, and other advantages of visual programming, in Chapter 7 (see pp. 227–231). Of particular relevance to

the matter at hand is the ease with which computational toys can be constructed in Max. Max provides interface tools for manually generating particular kinds of inputs and for displaying particular kinds of outputs. These tools can be created with a few clicks, and connected to an object or objects (representing a computational sub-medium) and arranged into a kind of control panel, or simple computational toy. [IT1] relates a specific example:

> *[The artist could] understand it in Max, pretty much… you should have a look at the patch, it was very difficult-to-understand patch, but you could see some things happening. We had it drawing, like a graph of the [accelerometer] readings, we could see stuff happening.*
>
> [IT1]

However, I can see scope for making this process even easier, and more attuned to changing creative needs, by having toys created semi-automatically. For example, in [IT1]'s description of a graph for accelerometer readings, there is presumably an 'accelerometer' sub-medium, which takes the raw data from the accelerometer device, resolves this into X,Y and Z acceleration components and double-integrates the accelerations to produce position information, and graphs that. Toys could be needed at any stage of such a process, in order to test the process, say, or to attempt to detect particular gestures with the accelerometer. Even with Max's toy-constructing tools at hand, it would be a tedious process to manually create and destroy interface components of toys at whim.

I recommend that technology designers implement a facility to have input and output interfaces appear as an 'interactive tool-tip' to every input and output of an object. My prototypical Max implementation , called gt.probe (developed for the *Cardiomorphologies v. 2* collaboration, Chapter 9) is shown here:



gt.probe in action. Moving the cursor over an outlet shows a temporary display of the value of that outlet. Moving the cursor away from an outlet hides the display. Clicking on elements of the temporary display makes them persist as standard Max objects.

The interactive tool-tip is a small interface for seeing or changing the value of an output or input to an object, and is displayed merely by rolling the mouse over the input or output concerned. Although my implementation is in a visual programming environment, it is conceivably possible to integrate similar functionality into a textual programming environment (both are recommended as ways to creatively engage with the computing medium). The 'interactive tooltips' are temporary modifications to the program. That is to say, they are actual program components, included in the program. If the mouse is moved away from the input or output to the object, the modifications are removed and the interactive tooltip disappears. However (in my implementation) if the user clicks on the tooltip, it becomes persistent—the modifications remain, and the user can incorporate the interface into a more permanent toy design.

Each Max object has the potential to know what kind of data it is producing, and what kind of data it accepts. At a simple level this is a data type, such as an integer value, floating point value, or a string, but more advanced data, such as "3D coordinates" or "video images" can be exploited too. This information can be used to automatically choose an appropriate interface for displaying or changing the data, though the user should have a way to change this representation (initially by choosing from a collection of appropriate automatic representations, but also by making it persistent and modifying it as part of the program).

We have established that the use of visual programming is useful to promote the construction of boundary objects. However, visual programming's use in interactive art collaboration has some side effects that are not necessarily conducive to engaging with the potential of the computing medium. In particular, one of the things that emerged was the focus on 'concrete', rather than 'abstract' programming when visual programming was used. In particular, the physicality of visual programming draws attention to the physical process of computing, rather than of the abstract structure which is computing's most unique feature (Chapter 5). [IT4] makes the point that this might not matter in interactive art:

> *Particularly once you're working in flat system-level patches with real-time response, it is more about control than program. You're not having to plan and make delayed decisions about building things up into final forms and ensuring they operate when everything is working together.*
>
> [IT4]

Yet designers should not discourage the possibility to build up 'flat' patches into abstractions and structures. Max, for example, has rudimentary support for building structure in terms of wrapping patches into single 'objects', and making many copies of these objects. Yet operations that are inherent in 'true' object oriented languages—classes, inheritance, polymorphism and abstract classes, are not available. If these concepts are given visual representations, the potential power of object orientation becomes 'physical', making the 'step up' from concrete operations to abstract structures easier to comprehend.

Designers of visual programming tools, in the context of a single universal computing system, should include these features, in order to satisfy Alan Kay's elegant concept of every object containing all the potential of a computer.

Technology can also assist with engagement in computation in a pedagogical role. The strongest preference for programming tools that was expressed by technologists in my study was for languages they had been taught, or that they knew how to get help using. Max's help was cited by [IT1] as a particular example, because every object has a help file that is an annotated Max patch, or toy for interacting with the object. [IT1] also liked PHP, since the online help was augmented with examples of programmers' code for particular purposes. [IT4] claims that the only way he has learnt about object orientation and other 'best-practise' programming techniques is by "seeing effective examples of good things with and without explicit pedagogical instructions" [IT4]. So the inclusion of effective programming examples is a strong recommendation for programming tool designers.

Smalltalk has a feature which allows the programmer to search for functions that produce specified outputs in response to particular inputs (for example, a programmer could search for all functions which produce the list "a, b" from an input of "b, a". Responses might include quicksort, swap, reverse)There is scope in future research for intelligent assistance with a programmers' process for finding help. To take an imaginary example, the structure of a part of a program could be compared with a database of program structures on the Internet, and similar structures could be presented to the user for comparison.

There were instances in my grounded theory study of both programmers and artists lacking the necessary domain knowledge to understand the mathematics of problems and assess potential solutions. In particular [IT1] had difficulty with the mathematics necessary to analyse accelerometer data, and an artist in the COSTART study thought that rendering 22^{21} images was feasible (it would take 150 billion billion gigabytes to store, and many years to render). While there is no replacement for a proper maths education, programming environments could benefit from rich, annotated mathematical libraries (and associated toys) for exploring relationships between and implications of numbers. Additionally, toys could be augmented with algorithm analysis (O(n), O(n^2), etc.), instruments to monitor processor usage, and loop execution time estimates, to help users get a feel for the potential power and capacity of computers.

On a more specific level, many of the technologists in my study expressed a need for tools to assist the management processes of working within larger teams, but which are conceivably useful in smaller teams too. Specifically, there were needs for tools to manage 'central repositories' of the design decisions, and for tools to record system behaviour, with annotations.

The use of central repositories of design decisions would allow those decisions to be captured in a less formal way than in traditional software engineering methodologies, yet in a more permanent and easier-to-keep-track-of way than in agile software development methodologies (where such decisions are written down on index cards and kept by the

programmer as a form of temporary contract between the client and the programmer):

> *I'm working on big projects, when there's say ten people working on it, I find that you have to have a central repository of decisions that have been made otherwise there's too many places where people can get the wrong impression of what their expectations are and simple things like email conversations, keeping track of those, or an Excel spreadsheet which has the expenditure on certain parts […], simple things like that are kind of touch stones*
>
> [IT3]

[IT4] expands on this concept in a multi-media sense, describing how the contributions of collaborators could be kept in the repository in an appropriate form:

> *there are many other aspects of processes, working particularly with larger teams, things which are really hard to capture, and there are probably tools that would support those more easily. Things like—even being able to record the output of interactive systems with annotations very easily and quickly, in a really direct way without having to go away and rig up external hardware or build lots of layers of software in tedious ways, things like that would be great. […] And being able to capture things on the other hand as part of the design process or as part of the observation process if you are trying to have audience evaluation […] there are lot of tools like that that I keep on thinking: Wow, that would be nice, I might one day think about developing to some extent. But in the context of single projects they are too far outside of those sorts of things to actually be justified.*
>
> [IT4]

Having stored design decisions in multimedia forms, there is little reason to prevent that media—such as sketches, graphics, sounds, journals or videos—from being included in the code as documentation of it. This is a kind of multimedia extension of Knuth's Literate Programming

(Knuth, 1984). As well as strengthening the documentation of the code, there is a potential artistic benefit. At one point in the COSTART project, an artist noted:"I got so bogged down in all the science part of it, I think I woke up one morning and went 'Is this art?"Transforming an artistic intent (or plain English) into code is not a reversible transformation, since information is bound to be lost. By including other creative media amongst the program code, the relationships between the code and the artistic intent can be made apparent, and loss is minimised.

[IT4] wishes to record the output of interactive systems (with annotation), in order to capture and demonstrate system behaviour in certain situations (for bug-tracking, and to observe particular scenarios of use). However, [IT3] claims "usually input is the hardest thing to manifest—output you can fake pretty quickly and that's not a problem". Relating this comment again to Rokeby's experience of relying too much upon himself to test his own artwork, it becomes apparent that, where input is hard to manifest, then some means to capture and annotate the input data of human audience members could help interactive art developers assess the range and vagaries of human response (in our collaboration, described in Chapter 9, George Khut tested the system by recording and replaying data logs of the heart rate and breath of several people).So capture and annotation of both input and output of computing media has uses. Relating this idea to the idea that computing media are made up of computing sub-media, and again with the idea that sub-media should be explorable, it makes sense to integrate data recording and playback as part of a toy's interface.

Summary

The recommendations were split into three areas, by order of precedence. Each area of recommendations assumes that the preceding areas' recommendations are also taken into account, but the recommendations should be somewhat useful even if taken alone.

Firstly, I recommended that a technology for allowing the computing medium to fully support creative engagement (described in Chapter 5) is developed. I envisage this technology as being fully object-oriented, with technologies borrowed from Smalltalk and Max/MSP to remove the conceptual, spatial and temporal aspects of the compiler wall.

Secondly, I used the findings from the grounded theory study (Chapters 6 and 7) to recommend methodological approaches for creating interactive art. Broadly, the approach recommended is that an artist attempts to specify, and a technologist attempts to construct, a meta-design—a tool for making the artwork, rather than the artwork itself. Traditional iteration can be used to build such a tool—there is a risk with intimate iteration that the technologist gets too caught up in the process of making the artwork itself. This might be fine, but it is likely to both increase the aesthetic influence that the technologist has on the art, and reduce the potential for the artist to directly engage with the computing medium.

Thirdly, I used the findings from both studies, and from the literature, to make recommendations for particular aspects of technology to support recommended methodologies, on a recommended computing system.

9. Study 3—Implementing and Evaluating the Recommendations

Introduction

The first study in this thesis (Chapter 5) was an examination of theoretical requirements of technology that supports creative work in general and interactive art making in particular. The second study (Chapters 6 and 7) was an examination of some actual methodologies used in interactive art making, to identify further implications for technology and recommendations for complementary methodologies. The findings of each were synthesised into a set of design recommendations for methodology and technology to support interactive art (Chapter 8). This study evaluates the recommendations by prototypically implementing them in the collaborative development of a new interactive artwork, *Cardiomorphologies v. 2*, and studying the results of the collaboration.

Description of the Artwork

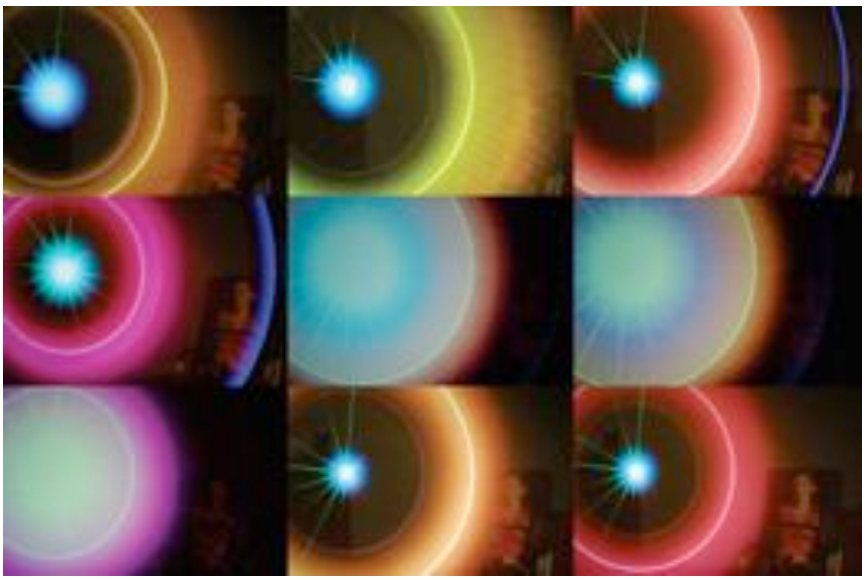


Figure 1. Photographs of the completed Cardiomorphologies v. 2. The participant is reflected by the screen.

Cardiomorphologies v. 2 (Figure 1) is part of a series of artworks by George Khut that use biofeedback technologies (Richards, 2005). Khut (whose biography is in Appendix 10 (on CD)) aims to give audiences an alternative insight into the way our thoughts and physical sensations are intimately linked. Specifically, he aims to enable participants to influence their heart rates through experimenting with their breathing and with their thoughts. Seeing and hearing their heart and breath helps the participants to perceive, focus on and monitor physiological changes which they may not have the ability to detect or recognise from sensation alone.

Individuals participating in *Cardiomorphologies v. 2* are seated on a comfortable reclining chair in a dimly lit and enclosed space. The participant is fitted with a breath (respiratory strain gauge) and heart beat (EKG) sensors. Graphics are displayed on a projected screen in front of the participant and a soundscape projects a sonification of the same cardio-respiratory data through a quadraphonic speaker array. The heart sensor produces a 'pulse' signal for every heart beat. The breath sensor produces a value which increases the more the sensor (strapped around the rib-cage) is stretched. These changes are analysed to produce various readings and statistics, the most important analyses focus on the subject's heart rate variations (HRV) and the relative amplitudes of the various

low frequency oscillations that constitute the moment-to-moment changes in heart rate (HRV) using an FFT based process of frequency analysis. The perception of changes in heart rate is fundamental to these works, especially with regard to the influence of breath on these patterns (Respiratory Sinus Arrhythmia).

The piece was a collaboration between George Khut, myself, and Lizzie Muller, a researcher into interactive engagement. This chapter focuses on the metadesign and development of the software of the piece. For a more (non-meta-) design- and audience-centred perspective, see Muller, Turner, Khut, & Edmonds (2006), which is included in Appendix 1.

Metadesign

> *Meta-design characterizes objectives, techniques, and processes for creating new media and environments that allow "owners of problems" (or end-users) to act as designers. A fundamental objective of meta-design is to create socio-technical environments that empower users to engage actively in the continuous development of systems rather than being restricted to the use of existing systems.*
>
> (Fischer et al., 2004, p. 4)

In this section I describe the metadesign, or the design of the environment for building *Cardiomorphologies v. 2*.

USE OF LEADING CURRENT TECHNOLOGY

Max/MSP (a.k.a. Max) is a data-flow programming language and environment, in which box-shaped 'objects' representing specific functions communicate with each other through connected inlets and outlets. The physicality of its objects has been shown to be useful in collaborations, because they are something to 'point at' which has computational meaning (Weakley, Johnston, Edmonds, & Turner, 2005) (see also Chapter 7). This is in itself a useful building block for toy-making, as, sliders can easily be attached to inlets, and outlets can easily be attached to displays, and so forth (there is room for improving this process, as I shall describe). Max also has no distinction between 'code-time' and 'run-time', meaning that the results of any change are immediately apparent,

which is also useful for collaborative development and saves time in incremental development.

We used Max/MSP to develop *Cardiomorphologies v. 2* because Max is useful for working with a wide range of sensors, as described previously. Additionally, George Khut has experience with it. In early talks between Khut and myself, Khut was willing to use another development system if the benefits outweighed the advantage of Khut's experience, but Max seemed sufficient, and was relevant to my research.

Cardiomorphologies v. 1 had elements written in both Max/MSP (used for the sound synthesis) and Java (used for the data analysis and graphics systems, programmed by another collaborator), and used third-party commercial software to analyse the heartbeat data. Since George was unfamiliar with Java, and the third-party software was limited and inflexible, one of the goals of the collaboration was to port as much functionality as possible to Max, to maximise Khut's level of control.

Max allows arrangements of objects, called 'patches' to be combined into a single patch object, similarly to functions in procedural languages. A programmer can double-click a patch object to 'lift the hood' and see the underlying arrangement of objects. This hood-lifting is hierarchical, and so allows the programmer to 'drill down' patches to the fundamental Max objects. This makes the interface a good candidate for a 'self-modifiable' computing system, as aimed-for above. However, the fundamental Max objects are quite high-level and non-modifiable. For example, there is no way to alter how a 'counter' or a video process works except by adjusting built-in parameters. Not only does this confound the innovative user who wishes to program at a deeper level, it also places higher demands on all users, who have to remember a larger number of more specialised objects. As Desain and Honing write, "the number of primitive [fundamental] objects needed to deal with special cases grows rapidly in languages that lack appropriate abstraction mechanisms; in Max this number is already above 100" (Desain & Honing, 1993).

Fundamental objects in Max are written in C, and the source is generally not available (Max's sibling, Pure Data, is open-source, but less mature (Puckette, 1996)). If a programmer wishes to create a new Max ob-

ject or to modify an existing object for which the source is available, he or she must immediately resort to programming in C, which is a complete paradigm break that withdraws most of the visual/toy-making advantages of the Max environment, and requires a wholly different skills-set of its programmers.

Max is not a true object-oriented system. It lacks inheritance, or polymorphism, and has a loose approach to encapsulation (there is no requirement for all external data for an object to be received through an inlet). It consequently lacks more refined features such as introspection, which would become useful for automatic computational toy-building.

In summary, whilst Max is the closest we have come in one system to a usable 'self-modifiable' technology, and assisted 'toy-making' methodology outlined above, it can only be used to test and demonstrate the usefulness of such technology and methodology through prototypes, hacks and with human intervention. Max cannot be used to completely implement 'self-modifiable' technology and automatically assist 'toy-making' methodology without being heavily rewritten. In the section below, "Future Technology", I describe a way to achieve the goals of real-world implementation of both of these.

ADDITIONS TO CURRENT TECHNOLOGY

Max, as a visual programming language has built-in support for non-automatic computational toy-making. In fact, every built-in Max object has a 'help' patch itself programmed in Max, which can be seen as a kind of computational toy. The overriding motivation for the toys that were created during this collaboration were to provide Khut with the knowledge to understand and use the objects comprising the "bit in between" the input and output. In *Cardiomorphologies v. 2*, the input signals are the heart pulses and the breath sensor strain, and the output is the graphical and audio displays. The "bit in between" was originally signal cleaning and analysis objects, but the meta-level of experimentation with and arrangement of signals became important during the collaboration.

I created toys using different methods, and at different levels of complexity, in order to get a feel for the implications that these variations might have. At the current level of research, one would expect to get a

validation of the grounded theory that toys are useful in interactive art development, but because of the sample size it is not realistic to expect to conclude that such-and-such a toy-making method has such-and-such an effect, but rather to be able to decide on parameters for future research. The simplest objects made in the collaboration, the "sensitive arithmetic toys" (Figure 2), were small variations of existing simple Max objects, written in C. (Max's arithmetic objects only perform the calculation when an value is received in the left-hand inlet, whereas the sensitive arithmetic objects calculate when a value is received in either inlet.) Only slightly more complex are the simple Max patches, created in the same way as in any other Max project, such as gt.round (Figure 3) and gt.limit (Figure 4), except they were supplied to Khut with a "toy" interface for experimenting with them.

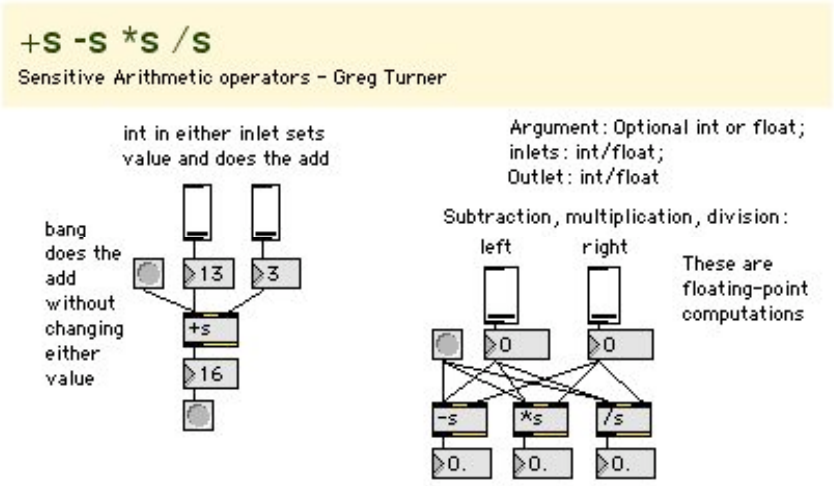


Figure 2. sensitive arithmetic toys

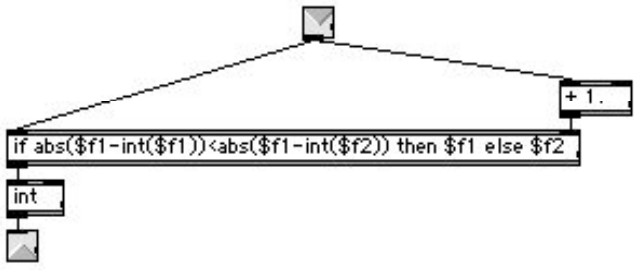


Figure 3. gt.round patch schematic

Figure 4. gt.limit patch schematic

More complex are objects which perform dynamic processes, such as the FFT object (Figure 5) and the gt.spring2 (Figure 6) and gt.Lspring2 (Figure 7), which spring-damp a value, or list of values. The FFT object was written in C for performance reasons, so could not be 'broken apart'; the spring2 objects were created in Max.

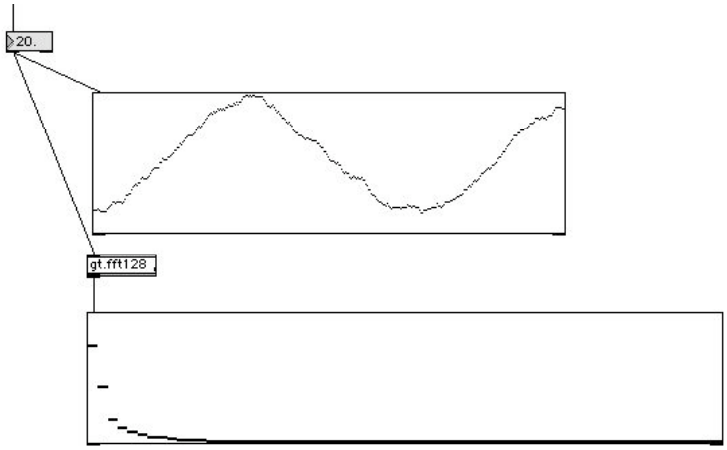


Figure 5. detail from gt.fft toy (the input signal simulates breath)

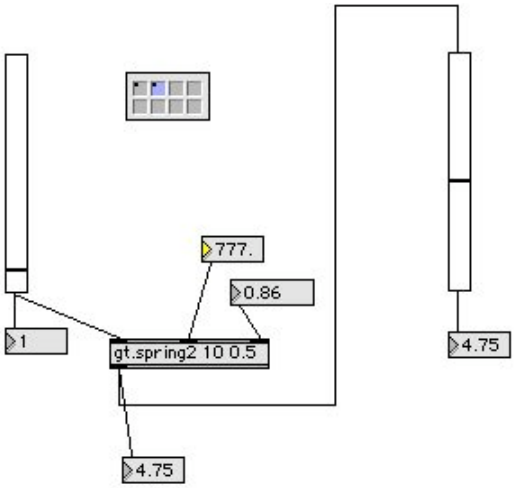


Figure 6. gt.spring2 toy

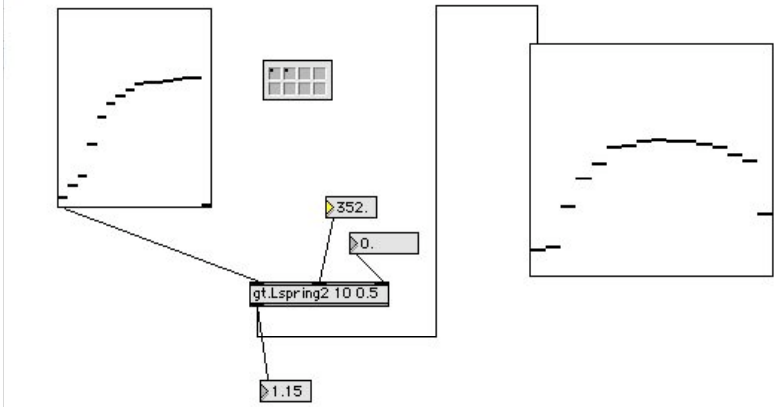


Figure 7. gt.Lspring2 toy

The toys' objects, and other existing Max objects were used to process the input signals. This resulted in a large list of about 50 variables representing different parts of the analysis. Examples, in OSC format, are:

```
/breath/raw_0_127
/breath/max_0_127
/breath/delta/inhale
/breath/delta/exhale
/breath/smoothed_delta
/heart/ibi_ms
```

```
/heart/ibi_min
/heart/ibi_normalized_0_1
/heart/fft_02
```

Figure 8 shows a simple, but large, toy for visualising and playing with all of the parameters for the 16 ring shapes of the display (as you can see, the display itself is too confusing to be used for identifying which rings have which parameters).

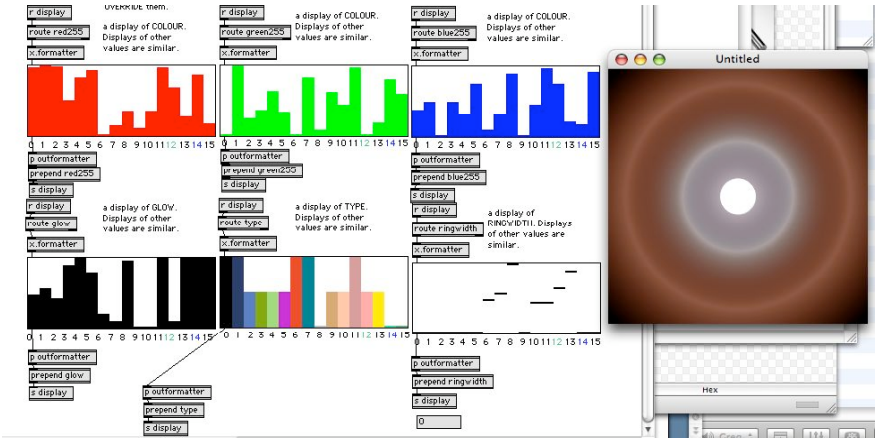


Figure 8. detail from the *Cardiomorphologies v. 2* display toy

The challenge became to map (and scale) the 50-odd analysis components to the 150-odd display parameters. Using patch cords in Max, this would have been a nightmare of spaghetti. Max uses "send" and "receive" objects to transmit data without patch cables, but these must be typed in by hand. The next pair of toys—one for sending messages to many possible places, one for receiving them from many possible places (Figure 9)—tried to resolve these issues by tracking all of the names of messages in the application and presenting them in a menu. This meant that Khut could not only choose what values to map to what display parameters, but he could store those choices in a preset to recall later.

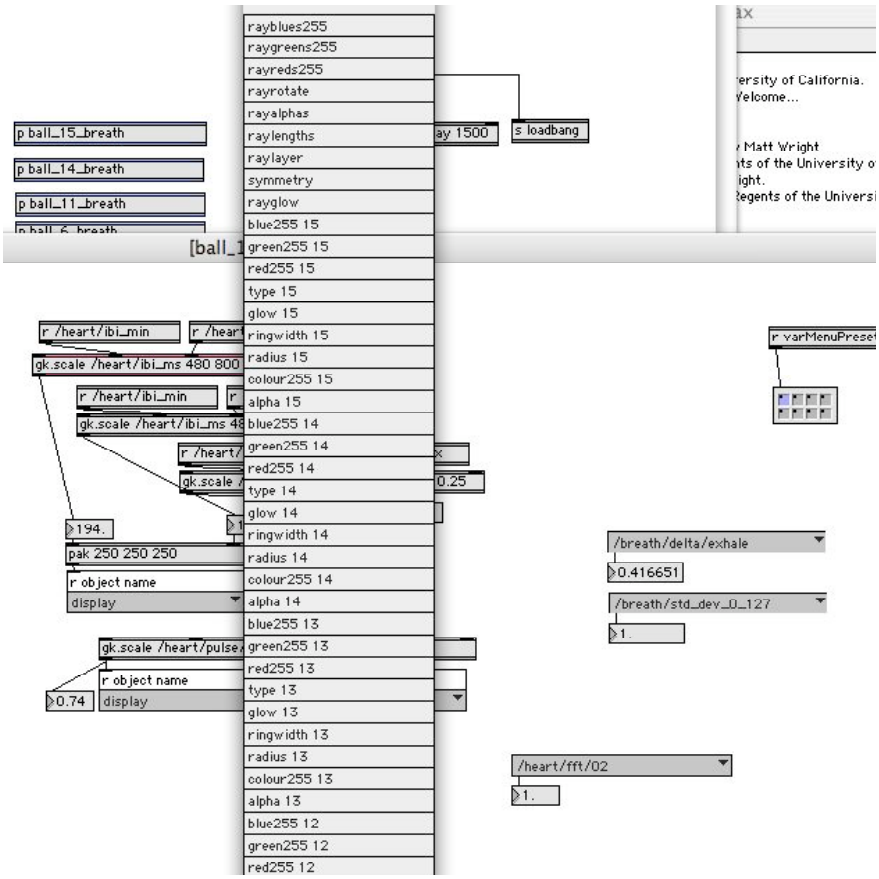


Figure 9. Message routing menus. On the left is a menu-driven "send" object. On the right are several menu-driven "receive" objects.

Finally, I created a prototype 'probe' tool for Max (Figure 10). After moving the mouse over an outlet of an object, a display appears showing the value of that outlet, in an appropriate form (numerical, counter, video windows, etc.) and using standard Max objects. This means that it is not necessary to manually create and remove processor-power-consuming displays when only a temporary reading is needed. Yet if a more persistent reading is needed, a click on a display element makes it persist—the Max objects comprising the display become part of the patch and can be manipulated in the usual way.

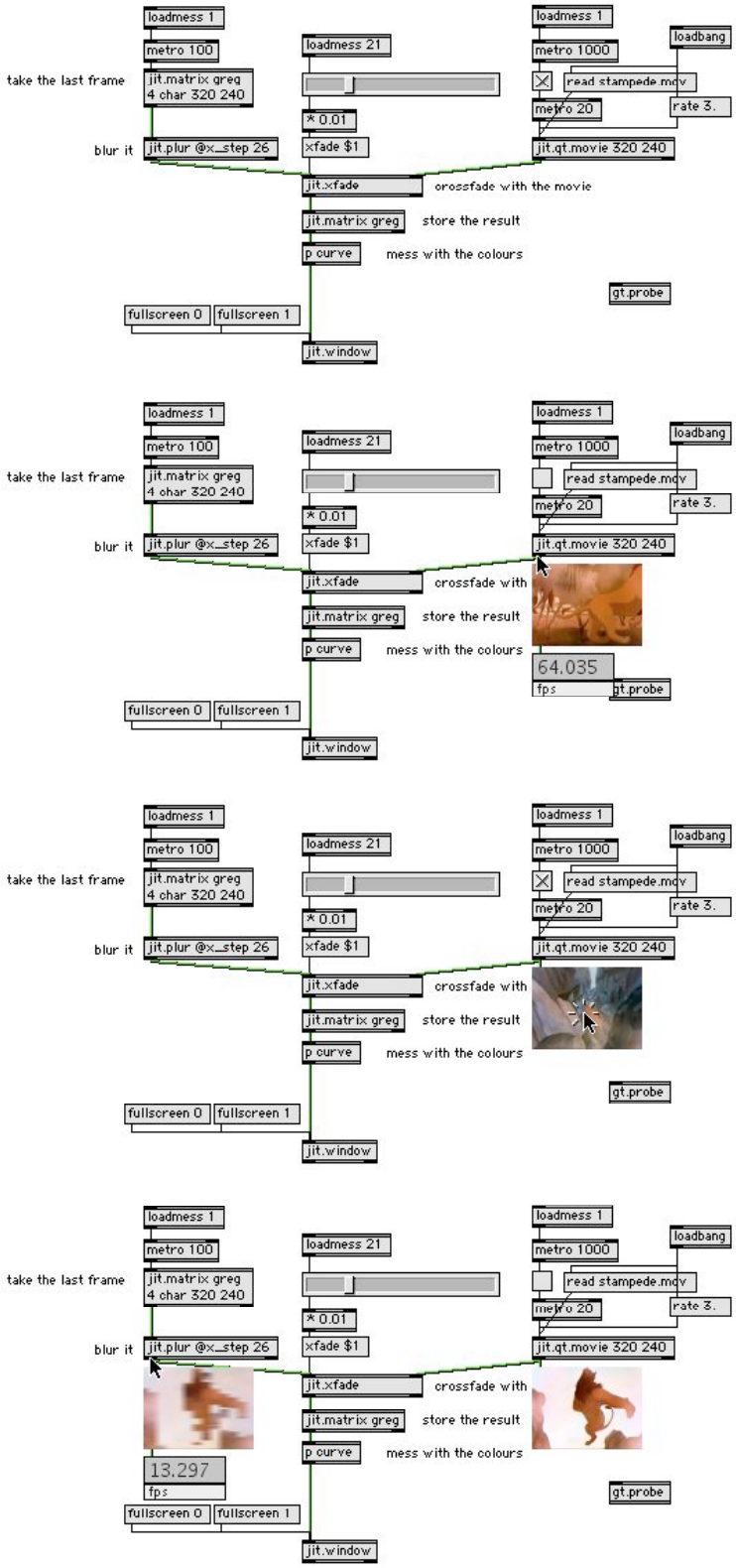


Figure 10. gt.probe in action. Moving the cursor over an outlet shows a temporary display of the value of that outlet. Moving the cursor away from an outlet hides the display. Clicking on elements of the temporary display makes them persist as standard Max objects.

Other toys and objects were made, but were similar to those presented above. One other object, created early in the collaboration, gt.score, was unfortunately lost shortly afterwards and has not yet been recreated (Figure 11) shows a mock-up of it). gt.score was a minimal graph of several values over time, in 2D and orthographic 3D, and was used as a test visualisation for *Cardiomorphologies v. 2* (Muller et al., 2006).

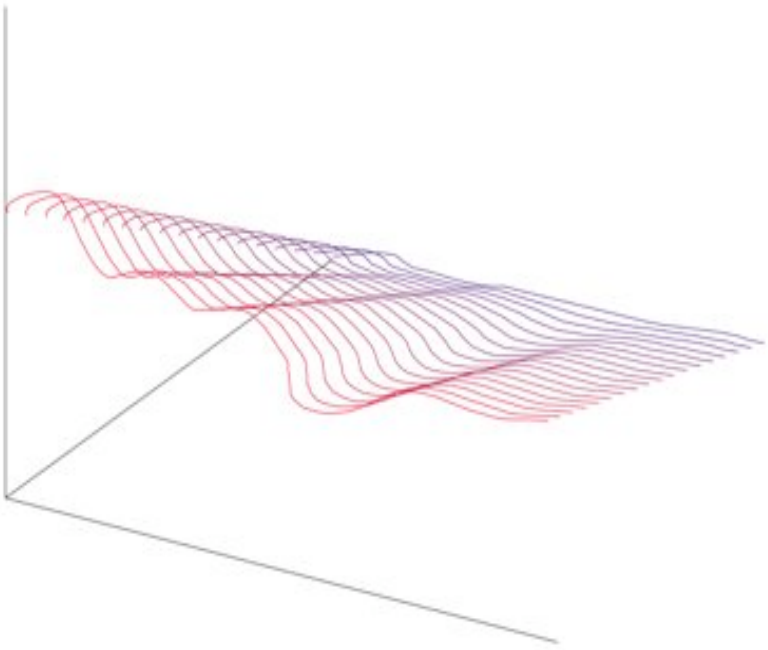


Figure 11. Mock-up of gt.score display

Evaluation and Discussion

As described in the methodology (Chapter 3), the artwork itself was evaluated by Lizzie Muller, using her methodology, which centres around the audience experience of the work as it develops. However, although the artwork was successful in its own right, this is only indirect support for the evaluation of the collaboration process itself, and particularly the new elements that I introduced, the computational toys and the computational-toy-making methodology. To that end, I also interviewed Khut about his view of my approach. Additionally, an excerpt from Khut's doctoral exegesis (Khut, 2006) appears in Appendix 9 (on

CD), with his permission, but I do not have ethical clearance to include it in my evaluation.

EVALUATION OF THE ARTWORK

The following is written by Lizzie Muller (with myself and George Khut as co-authors of other sections, not quoted here), and is an excerpt from the paper *Creating Affective Visualisations for a Physiologically Interactive Artwork*, presented at the IEEE Information Visualization Conference'06 in London, UK. A full copy of the paper, detailing the artwork's design and evaluation process, is in Appendix 1.

—Start of paper extract—

The experiences reported by the participants in the final version of *Cardiomorphologies* included intensely personal feelings and emotions, which demonstrated the affective power of the visualisations. Their accounts revealed the extent to which Khut's experiential aims of engagement and reflection had been achieved, and also some important insights into the role of richness and ambiguity in this success.

There was a great deal of evidence to suggest that the aims of *Quiet*, *Concentrated* and *Inwardly Attentive Focus*, and *Sensual* and *Kinaesthetic* were achieved. After a period of expectation at the start, all reported feeling relaxed and calm. The majority of participants reported that they were focused on their body and their breathing:

> *I'm just thinking about the heart and breathing calmly, that was the focus, nothing else*

Several participants reported the focusing and relaxing power of the image, for example:

> *My thoughts kept jumping to external thoughts but then I'd sort of get sucked back in and drawn back to the image, and then I just kept starting to relax more and more.*

Participants also gave indications that the visuals felt like a mirroring, or extension of their physiological and mental activities, for example at one point a participant is not sure whether they are causing the rings to move or whether the rings are causing them to breathe in a certain way. Interestingly most participants report that they notice changes in the

HRV spectrum rays depending on their mental focus, even when they don't fully understand what information the rays are displaying, a point that is discussed further [below].It is clear that the relationship between the visuals and the participants' mental and physical states leads them to a feeling of engagement.This supports previous research by Edmonds et al. (E. A. Edmonds et al., 2004) which proposes a set of requirements for successful engagement in physiologically driven computing and art. These include 1) ease of connection to input and 2) immersion in output, which together lead to 3) sense of personal connection with technologies or 'somatic extension'. In the case of *Cardiomorphologies* there is evidence that this state if 'somatic extension' was achieved, for example:

> *I'm feeling […] I don't know if it's the right word but integrated with the visual, because it's kind of synchronised with how you're thinking and breathing and moving and your heartbeat and everything all integrated together*

The majority of participants' experience accorded with the affective goal of *Explorative* and *Curious*.The opening sections of their report tend to be characterised by words such as "*curious*", "*fascinated*" and "*intrigued*".They explored the work in three ways; by experimenting with their breathing, with their thoughts or with trying to create certain patterns and shapes with the visuals. For most this process is described in positive terms, but there was one interesting exception. One participant describes a feeling of frustration at how little he could make the visuals do:

> *I thought "oh, well I'll try different things and see what I can make this thing do…I looked at the guy before me and I thought " oh he's not doing much maybe I can make it do all sorts of things" but I couldn't.*

This coincides with a point made by Edmonds et al that physiological artworks challenge our commonly held notion of control (E. A. Edmonds et al., 2004). Edmonds et al. suggest that it is more useful to think of the participant as "influencing" rather than "controlling" physiological systems, in which you are "being" rather than "doing" something in order to have an effect. It is an interesting challenge for the curatorial and interpretative aspects of such artworks to consider

how audiences might be encouraged to adopt this different approach to interaction, so as to avoid frustration.

Most participants seemed comfortable to adopt this alternative kind of interaction, and it facilitated a range of reflections about their body and life through both negative and positive images and a recognition of their physiological effect. There were two particularly interesting aspects to this, first that the participants all had only a very vague understanding of the kind of information being displayed by the spectral rays, and second that, nevertheless, the appearance of the rays prompted in most cases the recounting of very intimate and personally *meaningful* stories as a way of reflecting on the self.

Stories told by participants included thoughts of loved ones and of work, memories of holidays and reflections on health and previous or current bodily states. There was evidence of a strongly perceived correlation between these thoughts (and their associated physical sensations) and the visualisations, for example one participant says:

> *I tried to think of my girlfriend, who I haven't seen for three weeks. And I remember the lights were really going out when I was thinking of her"Another comments:"[I] kept trying to think of myself as feeling really good about my body, [imagining myself] on the bike, and just cycling along, and I noticed that I could see the effect of that sort of calmness, um, that came from being on my bike.*

This evidence supports Gaver et al's suggestions that ambiguity creates space for people to create meaning (Gaver, Beaver, & Benford, 2003). They are engaged in the visuals, but not fully immersed—there is a slight distance, created by ambiguity, which encourages active reflection, and requires that, to interpret the interaction, the participants build connections between their own physical and mental experiences and the visualisations they are being shown.

The aim of the action research project was to realise as far as possible the experiential and affective qualities desired by the artist. The results show that the final visualisations were very effective in this respect. Engagement was achieved not only by creating a close relationship between physical sensation and visual feedback, but also by adding richness

that encouraged creative exploration. Ambiguity played an important role in creating room for personal meaning making which encouraged a reflective experience. An important aspect of the affective impact of the visualisations is the attitude of the participant and a challenge for future work with physiologically interactive art is to encourage a mind-set where participants aren't trying to achieve a goal, but rather exploring and learning as they go.

—End of paper extract—

EVALUATION OF THE COLLABORATION PROCESS

I interviewed George Khut about the collaboration by on 19th May, 2006. The interview was semi-structured, in the same way as the grounded theory interviews were, and contained similar background questions for artists, with extra questions relating to Khut's concept of 'computational toys' and the toy-making methodology. Quotes from the interview are edited for readability here, but a verbatim transcript of the interview is in Appendix 8 (on CD). In summary, the findings from the interview and my reflection on the collaboration appeared to support the grounded theory, in that Khut expressed a preference for a toy-making methodology to a more conventional iterative development approach to understanding and attuning to the software, saying that it helped him "make sense of the data"—in other words, to understand how to understand the data. There was no new evidence to support self-modifiable computing systems, but existing evidence was corroborated. As mentioned before, Max already exhibits certain aspects of such computing systems, which were used as a matter of course.

Prior Approach Khut's approach prior to the collaboration was to attempt to give a specification to each programmer he worked with:

> *usually I am wanting them to help me in terms of analysing certain kinds of data or conditioning input from sensors to get certain kinds of data, so I say, I need this, this, this and this, and this is when I need it, this is how often I need it, you know, every millisecond or whatever … They generally go away and make the kind of object [that] does things for me, or a kind of application to do things for me, and we come back and look through it. And I usually say, yeah,*

> *that is great and go away and spend a few weeks working on and realising that, oh, no, there is certain thing doesn't work. (laugh). I have to call them back and say, oh, that thing doesn't do, it is not behaving the way I want it to or you misunderstood some aspects of it or it's inaccurate and therefore not so useful to me.*

In the grounded theory presented in the earlier study, I characterised his approach as 'traditional iteration'—making software through a slowly-iterating process of instruction, coding and evaluation. This is compared with 'intimate iteration', where the collaborators sit side-by-side and iterate development much more rapidly and granularly. One reason for the traditional iteration approach is that Khut seems to ask his programmers to carry out tasks that he is comfortable describing (e.g. Fast Fourier Transform), yet would not know how to build himself (e.g. "I struggle with mathematics"). Khut prefers to do by himself the tasks that he has difficulty describing, through experimentation and play.

> *Actually mapping those different kinds of the data to the art objects I do myself. So decisions about the kind of colour or size of things, or the sound texture I make myself. But the functionality in terms of how you analyse incoming data and how you might manipulate the information in between, the programmer works on.*

Much of Khut's specification of software is given by example. For instance, rather than specify the format and range of the heartbeat and breath data, he uses Max to supply a sample archive of that data. This has the advantage to Khut that he knows he can supply the data in a similar format, and to the programmer of easy integration. There is a parallel here with the physicality of Max's programming objects, in that there is a definite artist preference for actual, rather than abstract or metadata. On the other hand, Khut uses sample input data to test the program's functionality, which means that the program is built for that sample data only. In a gallery, members of the public may produce much more varied data.

> *it's only when you are in a real world situation… maybe I am not reproducing those variables in my studio, that a lot of things become apparent. Once [the piece] gets into a gallery and you've had it*

> *running for a day, you will see a variety of different users come through. That's when it comes a lot of [design] decisions are revised.*

This is a kind of converse of David Rokeby's observation (Rokeby, 1998) that interactive artists are prone to modify their behaviour in response to their over familiarity with the piece. The modified behaviour turns out not to be natural for unfamiliar audiences.

Feedback on Toys When I asked what his concept of computational toys was, Khut contrasted 'toys' with 'tools', but did not draw a particular distinction. He described tools as "general", "objects", "utilities", such as the spring objects and the FFT objects, and toys as objects that "could be put together to create different kinds of things". Khut described the graphic rendering patch in particular as conveying a "sense of play". This indicates a misinterpretation of my concept of toys (any process or algorithm with an attached interface), which could have come about through the straightforward design of many of the toys, and their relative prevalence in the Max/MSP environment. Khut agreed that his perception could be to do with the flexibility of the objects—the spring and FFT objects had zero to two parameters; the display toy had 160-odd.

Khut's reported enthusiasm for the toys seems to be directly related to how reliably they worked, as opposed to how complex they were, or how useful they were for the piece. The probe toy in particular did not work reliably enough on all objects (because of the technical limitations of implementing something like this in Max). The message routing tool worked well as an exploration and experimentation toy, but Khut replaced the menu selections with hard-coded routing, because the settings would occasionally not be stored.

Overall, Khut expressed a definite preference for toy-making methodology, and is using it in subsequent collaborations:

> *in subsequent collaboration with [another programmer], we just decided that that's a much tidier way of collaborating, that they make a toy with all of these variable inputs, and then I decide how I want to get the data to those things. Whereas previously, I'd said I'm gonna give you this data and it's got this range and it's going to*

> *respond in these ways and you make a thing which is based on that. He'll now just say, well, here is the message. You can feed the values to those messages in these bounds, you can decide the responsiveness to it and all those kind of aesthetic things.*

Khut sees toy-making as a slower methodology, but says the slowness comes from the time it takes him to get familiar with the toys. However, the time it currently takes to create a toy manually with Max could be radically shortened, e.g. with a more complete and robust gt.probe.

However, Khut still feels "frustrated with [his] degree of mastery over [the] processes of making interactive art"—it is difficult for him to know when an interactive piece is finished, compared with a non-interactive one. My hypothesis is that this is due to a number of factors. Despite his improved ability to arrange data appropriately during this collaboration, Khut wishes for more foundational understanding of the processes guiding the data. But even with this, a "finished" interactive work is impossible without a contribution from participants, each of whom interacts with the work in a different way.

Summary

Cardiomorphologies v. 2 was developed as a collaboration between George Khut, myself and Lizzie Muller.

The metadesign for *Cardiomorphologies v. 2* was constructed as a traditional iteration, with toy-making explicitly included. In constructing the metadesign, I decided to use a combination of existing technology that aligned with some of my recommendations, and to use that to build prototypes of new technology that implemented others of my recommendations. The existing technology identified was Max/MSP, since it has the strengths of visual programming languages that were found to be so important in interdisciplinary collaborations (Chapters 6 and 7).

Lizzie Muller evaluated the artwork and found that it aligned with the artists' aims, which can be interpreted to mean that the aesthetic influence of the technologist was minimal and that the metadesign was successful in giving George 'designership' of the *Cardiomorphologies v. 2* artwork.

I interviewed George Khut about his views on working with my meta-design. Overall, Khut expressed a definite preference for a toy-making methodology, and is using it in subsequent collaborations.

10. Conclusions

Summary of Argument

As we use software to explore new areas of knowledge, the problem is not how to make the software, but more how to know what software to make. In contrast to conventional software methodology, in creative, innovative software we cannot know what the requirements truly are until the software is finished. Many software engineers are used to the idea that when software made in this traditional way is finished, the requirements have changed. My work is centred on finding tools and techniques to make software that has few formal requirements, and many requirements that cannot be directly translated into a programming language. But what does this mean? In the introduction (Chapter 1) I introduced a Venn diagram of a notional creativity support environment in order to structure a research question using it. My research question is:

Q: **In a situation which otherwise supports creativity, what would be useful technology and related methodology to help people to make interactive art?**

I approached this question from various angles. Firstly, I needed to understand what is meant by "creativity support" and by "interactive art", from technological and methodological perspectives. Secondly I used this understanding to recommend supportive technology and methodology. Thirdly I evaluated my recommendations by implementing this support in a new interactive art project.

Having introduced the topic of research, I fleshed out its four main themes in the Literature Review (Chapter 2). Firstly, I reviewed the topic of creativity, since without doing so the concept is prone to being treated vaguely. For this research I became particularly interested in Margaret Boden's notion of *conceptual spaces* (e.g. Boden, 1994). Conceptual spaces can be explored or transformed in order to produce radically original ideas. As exploration takes place, the constraints of the space become apparent. It follows that if constraints can be transformed when needed, those constraints support creativity at those boundaries. It also follows that if constraints cannot be transformed when needed, creativity is limited by these boundaries.

Secondly I reviewed creativity support qualities, which are qualities of situations, minds, artefacts and activities that researchers say are supportive of creativity. These qualities were arranged on the Venn diagram introduced in Chapter 1, as qualities of situations, minds, artefacts and bodies, though there are overlaps between these. Qualities of situations include "peaceful setting", "adequate resources", and "synergistic extrinsic motivation". Qualities of minds include "intrinsic motivation", "creativity-relevant skills", "collaboration skills" (at conflict with "independence"), "self-confidence" and "synaesthesia". Qualities of artefacts include "simplicity", "externalisations of knowledge" and "stimulation of emotion". Qualities of activities include "problem-finding", "problem-solving", "collaboration" and "iteration".

Thirdly I will review the history of interactive art, and the styles and technologies used in interactive art today. What do interactive artists want to do with computers? Anything! Interactive art stems from a rich

history of involving the audience in art. It represents some of the broadest and most demanding uses of computers, due to the unusual juxtapositions of hardware and software, the realtime response it often requires, and the variety of forms it can take. Opinions vary on whether or not interactive art must be programmed by the artist. A moderate view, adopted in this thesis, is that artist need not program if the artwork does not involve computing *per se*.

In anticipation of using it in discussing technological support for programming interactive art (Chapters 5, 7 and 8), I lastly reviewed general principles and styles of current End-User Programming (EUP) research. In summary, metadesign is the concept of designing (tools) for designers. End-User Programming (EUP) is a type of metadesign, and comprises many approaches. Perhaps the largest approaches are technological (making programming easier) and pedagogical (widening the culture of programming), though many projects combine both. Of the technological approaches, I distinguished between intuitional and expositional paradigms, and argued that intuitional paradigms could inhibit creativity because they override subjective interpretation of the programmer's intent.

My methodological approach to creating new knowledge (fully described in Chapter 3) was as follows. The first thing to do was to explore tools—to become well-acquainted with existing tools for making interactive art, and their creativity support features. This process was similar to a critical literature review, but with the inclusion of an argument, based upon that literature, for certain needs of technology. The second thing to do was to explore art-methodology in collaborations. Since this was an exploration of a social process, without imposing predefined goals, I needed to use a research method that would provide me with the art-methodological needs of such collaborations. Grounded theory analysis satisfied these requirements. Thirdly, I wanted to use the tools expertise gained in the first stage with the art-methodology expertise gained in the second to recommend useful technologies and related art-methodologies. Finally, to evaluate the recommendations, they need to be used in the context of actual art collaborations. I aimed to participate in one such collaboration, involving prototypes of the recommended technologies and art-methodologies. I intended to observe

the collaboration and to interview the artist to understand the improvements and opportunities for further research.

The preliminary studies chapter describes two art collaborations I have been involved with which influenced the current work. The first collaboration, *cubeLife* with Dave Everitt, is my first artwork, and inspired me to do a Ph.D. in this area. The second collaboration, *Séa.nce* with Norie Neumark and Maria Miranda, is interestingly self-referential because it was a collaboration across geographical distance, to make an artwork which itself deals playfully with collaboration over geographical distance. *cubeLife* highlighted a need to more easily develop exploratory interfaces in Java. *Séa.nce* is an example of an interactive art development process that quickly came up against the limitations of its development environment. My descriptions of these pieces were echoed in the chapters which followed, as technologists described coming up against similar limitations and requiring similar features of technology.

Before I went on to study interactive art collaborations, or make recommendations, I needed to figure out what creative potential computing had, that useful technology and methodology should support. In Chapter 5 I argued that as our conceptual spaces change through creative activity, so too do our requirements of the tools we need to support our conceptual spaces and creativity. If our tools are software, then changing software is the best way to keep tools in pace with our changing conceptual spaces. Yet we are usually prevented from changing software—not only because programming is hard, but also because software is compiled, closed, rigid, static, when it could be de-compilable, open, flexible, dynamic. Today we are given a world where programming and using, compiling and runtime are entirely separate, exclusive situations, as characterised by technology, and the way we are supposed think about technology, as though it was fundamental, or innate, or natural, or necessary. But if that separation is truly fundamental to computing, there would exist no systems that didn't enforce it. Therefore, if any system can be found which does not enforce such a separation, then that separation would not be fundamental, but rather a particular mythology. I have shown examples of these systems.

While the separation reflects particular pragmatic concerns in software design, these concerns need not dominate as they do now. The separation of programming and using is at the heart of a false dilemma in contemporary computing, and it is critical to overcome that false dilemma in order for computing to truly support creativity in accordance with its potential.

My second study (Chapters 6 and 7) was a grounded theory study of the roles of technologists in interactive art collaborations. It began with a lengthy description of the coding and analysis process of the CO-START and COSTART 2 projects, which produced a table of categories relating to interactive art collaborations. Subsequent iterations of coding and analysis were based on interviews I conducted with artists and technologists about the issues raised by the coding so far. Having coded the interview transcripts, the coding was incorporated into the coding table produced by the COSTART coding, to produce the complete coding table, with categories in a hierarchical arrangement, quotes from the interviews, and memos to assist my drawing the categories together into the theory. The write-up of that table, incorporating quotes and memos is the grounded theory analysis.

The theory contends that of fundamental importance is the process of attuning between humans, and between humans and technology. In particular the technologist must ideally reach a holistic understanding of the art system across many continua: from artistic concept to technological implementation; from front end to back end; from people to technology; from gentle programming to deep programming. The technologist bears the brunt of the responsibility for attuning the sides of these continua, and, poorly managed this can easily result in frustration or powerlessness on the part of the non-programming artist. Technological development of art systems starts with a metaphorical description from the artist, which is embodied in successively deeper computing systems by an attuned technologist. The technologist also attunes the computer to the artist, through a combination of 'intimate iteration' and (preferably, for creativity support purposes) 'computational toy-making'. Ideally the attuning happens to the extent that the artist no longer needs the technologist, yet this does not always happen in practice be-

cause of the extra work involved in making toys and the lack of specification of them.

'Playing' with technological 'toys' is crucial to the development of interactive art systems, for these reasons, most of which are particular forms of attuning:

1. finding the rules that govern the behaviour of the algorithm,
2. developing the 'tendencies' or character of the system,
3. intuitively learning the 'language' and capabilities of the algorithm by:
    a. exposing the potentials of the algorithm's dynamics,
    b. making the algorithm's place within the system apparent—highlighting the structural interactions and limitations
    c. providing a realtime response to artists' actions
4. producing computational, technological and artistic meaning simultaneously (which reduces ambiguity, produces successful boundary objects, and so assists interdisciplinary attuning),
5. making the artist feel more comfortable and empowered,
6. incorporating synthetical and analytical approaches to problem solving, and
7. taking the technologist's interpretation out of the loop.

These toys, then, are boundary objects that embody a language for creating meaning between artist and technologist, and artist and computer, and perhaps less necessarily, technologist and computer. (There is also scope for these toys to become boundary objects for other communities, such as audience members, other artists and technologists, researchers and curators).

The theory has given us a new way to understand artist–technologist collaborations as dynamic processes of attuning through social relations, the technologist's role, and through technology. It remains for us to find

ways to exploit this new understanding in the context of technology that supports computing as a creative medium, and Chapter 8, the design recommendations, was a first step on that path.

The recommendations were split into three areas, by order of precedence. Each area of recommendations assumes that the preceding areas' recommendations are also taken into account, but the recommendations should be somewhat useful even if taken alone.

Firstly, I recommended that a technology for allowing the computing medium to fully support creative engagement (described in Chapter 5) is developed. I envisage this technology as being fully object-oriented, with technologies borrowed from Smalltalk and Max/MSP to remove the conceptual, spatial and temporal aspects of the compiler wall.

Secondly, I used the findings from the grounded theory study (Chapters 6 and 7) to recommend methodological approaches for creating interactive art. Broadly, the approach recommended is that an artist attempts to specify, and a technologist attempts to construct, a meta-design—a tool for making the artwork, rather than the artwork itself. Traditional iteration can be used to build such a tool—there is a risk with intimate iteration that the technologist gets too caught up in the process of making the artwork itself. This might be fine, but it is likely to both increase the aesthetic influence that the technologist has on the art, and reduce the potential for the artist to directly engage with the computing medium.

Thirdly, I used the findings from both studies, and from the literature, to make recommendations for particular aspects of technologies to support recommended methodologies, on a recommended computing system.

The final study (Chapter 9) evaluated my recommendations by prototypically implementing them in the collaborative development of a new interactive artwork, *Cardiomorphologies v. 2*, and studying the results of the collaboration.

Cardiomorphologies v. 2 was developed as a collaboration between George Khut, myself and Lizzie Muller. The metadesign for *Cardiomorphologies v. 2* was constructed as a traditional iteration, with toy-making explicitly

included. In constructing the metadesign, I decided to use a combination of existing technology that aligned with some of my recommendations, and to use that to build prototypes of new technology that implemented others of my recommendations. The existing technology identified was Max/MSP, since it has the strengths of visual programming languages that were found to be so important in interdisciplinary collaborations (Chapters 6 and 7).

Lizzie Muller evaluated the artwork and found that it aligned with the artists' aims, which can be interpreted to mean that the aesthetic influence of the technologist was minimal and that the metadesign was successful in giving George 'designership' of the *Cardiomorphologies v. 2* artwork.

I interviewed George Khut about his views on working with my metadesign. Overall, Khut expressed a definite preference for a toy-making methodology, and is using it in subsequent collaborations.

Situating the Research

KAUTZ ET AL.'S CROUNDED THEORY STUDY OF PROGRAMMING

We can draw some comparisons between the commercial software development in Kautz et al.'s grounded theory study (Kautz, 2004; Kautz et al., 2004) and the non-commercial software development in mine. The Kautz study identified five categories which affected the practical use of systems development methodologies: *universality* (there is no universally applicable methodology), *confidence* (in the work's progress as shown by the methodology), *experience* (means a technologist is more likely to pick and choose specific techniques from methodologies rather than follow them all), *co-determination* (choosing the methodology created a feeling of responsibility to use it) and *introduction* (providing appropriate training in and transition to a methodology).

Except for in one of the cases in my study, I found no cases of plan-centric (or 'engineering-style') software development methodologies being used. (The exception was in the case where the artist was able to specify precisely what he wanted, and a requirements analysis and UML diagrams and so forth were produced.)

I found Kautz et al.'s categories to be in alignment with this study, but the small scale of the communities, and the high expertise and autonomy of the technologists means that *introduction* category becomes virtually irrelevant, and conversely that the *expertise* category becomes highly relevant. The confidence category is also diminished in importance, because the artists (the equivalent of customers or managers) are closely involved in the development process, and trust of technologists is valued. *Co-determination* becomes sometimes inaptly named, as usually the technologist is the only individual with a stake in the methodology choice. In the large collaboration I studied, *co-determination* seems an important factor in methodology adoption by individuals concerned, but the details of this would require further investigation. *Universality* is interesting. The approaches taken by technologists working with artists could all (with the single exception mentioned above) be characterised as being forms of 'agile programming', and indeed a form of agile programming was explicitly adopted as the methodology for the large project in the study. One of the artists from that project said: "I continue to be attracted to the philosophies and some of the techniques of agile programming, principally because I see it as a means of operating whereby you do have a holistic understanding of the eventual delivery but you address immediate and evolving needs rather than fantasising that you can describe a project in the brief right at the start". Agile methodologies are characterised by their ability to accept changing situations, rather than trying to predict what they might be. They also tend to place themselves in subordination to the people using them, rather than being processes or frameworks in which people must operate. These things make it eminently suitable for developing artwork, since the technological requirements are often imprecise throughout the project, and also that part of creative practice is breaking with the convention of processes and frameworks. However, agile development methodologies used in isolation would result more in the 'intimate iteration' style of development, identified in this study to be limiting in terms of artist engagement with the computing medium.

HEURISTIC EVALUATION OF DESIGN RECOMMENDATIONS

Though I have evaluated prototypes of my design recommendations in an art collaboration (Chapter 9), let us also briefly examine the design

recommendations given in Chapter 8 against Mitchell Resnick's list of "Design Principles for Tools to Support Creative Thinking" (M. Resnick et al., 2005), which we first encountered in the literature review (pp. 49-50). Remember, as Brock Craft explains (Craft & Cairns), merely adhering to a set of guidelines does not automatically make a design worthwhile, yet, having also evaluated the recommendations in the real world, I want to use this example to make clear the alignment of my research with the literature. Resnick's list follows, then, with my notes about the alignment of my research:

1. *Support exploration:* The very motivation of my work has been to support creative exploration. The computing medium inherently supports users' creative exploration of it, yet the prevailing mythologies about compilers work against such exploration. My research has identified ways to overcome these mythologies by implementing a single, universal computing system.

2. *Low threshold, high ceiling, and wide walls:* These are metaphorical descriptions of cognitive barriers to entry, and limitations of a particular creativity support tool. I claim that my work has potential for the lowest threshold, highest ceiling and widest walls possible in the computing medium—the threshold can be as low as any interface to any system we are capable of developing today, and the ceiling and walls are as high and wide as the computing medium itself. This is because using a single universal computing system with a) an appropriately self-similar structure, and b) an appropriately well-designed interface for "lifting the hood" to explore and change underlying processes, would allow people to gradually transition from gentle programming (interface manipulations) to deeper programming (underlying commands and structures).

3. *Support many paths and many styles:* This is potentially a point of my research that requires further development. I recommended a hybrid text and graphical styles of programming, the two most dominant paradigms in programming, to cover the biggest range of tastes and to get 'the best of both worlds'. On the other hand a single universal computing system (as I have recommended) must

remain single if it is to be universal. Having said that, the single universal computing system I have recommended allows its own modification, so people could potentially modify it to suit their own styles.

4. *Support collaboration:* My grounded theory study of collaboration in interactive art has ensured that at least some needs of art-technology collaboration are taken into account. Treating artists as quintessential creative workers, it is reasonable to suppose that my findings apply to wider forms of creative collaboration, though this will need to be tested. However art-technology collaborations have wider needs than of programming technology, and other creative collaborations will have other needs still. Some suggestions are made in the Future Work section (see p. 291).

5. *Support open interchange* [Seamless operation with other tools that the creative user may wish to employ]: Hypothetically, using a single universal computing system will guarantee that two subsystems within it are able to operate together using any conceived technique. However, other software already exists outside of such a system, which is likely to be less flexible. Since a single universal computing system would allow a computer to do anything it is capable of doing, then if a computer is capable of cooperating with such a piece of software (i.e. if it is not encrypted, for example), then it could be made to do so.

6. *Make it as simple as possible—and maybe even simpler:* One of the primary reasons to adopt Alan Kay's version of object orientation—that each object encapsulated the entire possibilities of the computer (including other objects) (see p. 163)—was its simplicity. One need only figure out how any one of these objects works in order to figure out how any other one works. As for the simplicity of interfaces for exploring and manipulating these objects, I have made design recommendations which are intended to reflect the simplicity of Kay's approach, but I have left the implementation of these recommendations for the future, so their simplicity or otherwise remains to be seen.

7. *Choose black boxes carefully:* The black boxes of a computing system which fit with my design recommendations would be the primitive elements of a fully object-oriented computing system—the primitive elements of computing itself. The limitations of these black boxes are literally minimal.

8. *Invent things that you would enjoy using yourself:* I do, and have enjoyed using the things that I have invented.

9. *Balance user suggestions with observation and participatory processes:* I have done what could be the meta-meta-design (see point 11) equivalent. I have balanced my own suggestions (in Chapter 4) and argument (in Chapter 5) with a process of observing others (Chapters 6 and 7). In that latter process, I balanced (admittedly second-hand) observation of artists and technologists with questions designed to elicit their own suggestions. Further I have involved George Khut in the participatory process of specifying the requirements of the metadesign for *Cardiomorphologies v. 2* (Chapter 9).

10. *Iterate, iterate, then iterate again:* My grounded theory study touched upon two varieties of iteration: traditional iteration and intimate iteration. Traditional iteration is well established as a sensible means for refining a software design or metadesign. My contention, however, is that in intimate iteration, the technologist becomes too caught up in making the artwork to be able to make tools for the artist. This is not always a bad thing, but in the case of the toy-making metadesign that I recommend, it is, because it denies the artist opportunities to become more directly engaged with computing or with the artwork's manifestation if he or she wishes.

11. *Design for designers:* In Chapter 9, the *Cardiomorphologies v. 2* collaboration, I did exactly this. Yet the majority of this research has been *about* design for designers. If Fischer calls design for designers 'meta-design', then I had better call Resnick's and my own research 'meta-meta-design'.

12. *Evaluation of tools:* Resnick writes, "it is still an open question how to measure the extent to which a tool fosters creative thinking." (M. Resnick et al., 2005, p. 34). I agree, and my research has not attempted to close this question. As Resnick recommends, I used longitudinal studies to discover and evaluate my recommendations.

Future Work

My research has highlighted particular areas for future work. There are

METHODOLOGICAL IMPLICATIONS

As recommended in the NSF Creativity Support Tools report (see p. 93 of this thesis), I have used a variety of methods to produce and evaluate my design recommendations for making interactive art. This variety has produced a rich understanding of this particular creative environment, particularly through grounded theory.

It remains to be seen whether this or similar research methodologies could be brought to bear on other creativity-technology situations, though it is my belief that it could be very successful. One potential advantage of using grounded theory is that each grounded theory study could be compared in the same way as categories and codes are compared in a single study, and built up into a 'super' grounded theory of creativity support.

END-USER PROGRAMMING AND CREATIVITY SUPPORT

My approach to making computing media support creative engagement is incompatible with the 'intuitional' approach to end-user programming because the complexity of the necessary AI would undermine the simplicity of computational media on its own terms (see pp. 74–77) for an expanded reasoning). However, I believe there is ample scope for AI assistance in trying to understand what a user is trying to achieve pointing out similar existing tools, or programs, or code.

TEACHING PROGRAMMING

Although my recommendations should result in easier access to programming, by exploiting familiarity of interaction with a rich computing underpinnings, there is still work for our pedagogues, in terms of

deciding whether, or how to integrate programming education into formal curricula.

For example, I believe that there is no reason why computing should not be taught in art schools, rather than engineering or technology schools only. For that to become the case, we need to rethink programming to support artistic creativity as well as scientific creativity. I believe this thesis is a significant step along that path.

An additional open question, suggested by an examiner of this thesis, is the question of how much and what aspects of technology an artist should learn in order to take advantage of its full potential. I feel as though, so long as the technology supports its own exploration, down to a fundamental level, an artist need only be shown how to conduct this exploration.

IMPLEMENTING MY RECOMMENDATIONS FURTHER

I have evaluated my recommendations in the context of one new, in-depth collaboration. It remains to be seen the extent to which my recommendations are applicable in other collaborations, and for what reasons. For example, some art-technology collaborations may have very easy to specify technology, in which case toy-making would not be necessary. There may be different personalities of artists and technologists which I haven't come across. Studying a wider variety of collaborations would add richness to design recommendations for supportive methodology and technology.

One way to promote toy-making methodologies would be to publish a handbook for effective artist-technologist collaboration. The beginnings of a list of guidelines for artists and technologists follows:

Artists:

- Choose technologists wisely. You are likely to have a better experience with an accomplished technologist who doesn't know how to do what you ask, than an expert technologist who thinks he or she knows exactly what you want.

- Technologists may have technological motivations for making suggestions, without necessarily being aware of a negative effect

on the aesthetics. If a technologist suggests a way forward that doesn't sound ideal, ask why.

- If you can't specify an artwork, specify the *tool* for making an artwork, which is easier (but could be more work on the technologist's part). Using a tool will give you greater self-sufficiency and control over the final result.

Technologists:

- Check your own motivations for making aesthetic choices and suggestions: are you really putting the artwork at the fore, or are you making less work for yourself?

- Be professional. However, this is not about swallowing your pride and always doing what you are asked. Artists demand engagement, and engagement can make you a better problem-solver.

- Don't be reluctant to try something that couldn't possibly work, if there's time. Showing the artist what happens will build trust and educate the artist in the possibilities of technology.

- Gather skills by experiencing a wide range of artworks [as [IT3] did]. Try to find out how other artworks are made. Similarly, share your own code or techniques.

ARTISTS WHO PROGRAM

It would be interesting to attempt to study artists who *can* program, and their relationship with technology. Is their artistic vision more concrete or fully-formed at the outset? Do they have a clearer idea of the path to create the artwork? Do they construct toys for themselves? Is there a preference for visual or textual programming?

SKETCHING RECONSIDERED?

[IT1] reported programming to be akin to the process of sketching, in that it was often possible to discover something new in the sketch. This is strongly reminiscent of Donald Schön's theory of reflection-in-action (Schön, 1983).

Traditionally, sketching takes place in a different, lightweight, medium to the finished artefact. For example, presumably the sculptor Frédéric Bartholdi, when commissioned to design the Statue of Liberty, make sketches on paper, and in clay, before building a model for engineers to convert into the massive structure which stands in New York Harbor today.

In contrast, with my conception of a single universal computing system, using Alan Kay's object orientation, it becomes possible to represent the entire possibilities of a computing media in any part of it. This could mean that a program could be transformed from a sketch directly into a finished product without being re-written in a different media.

Of course, this begs the question of how such a programmatic sketching system might be designed or behave.

CURATORIAL IMPORTANCE OF ART DEVELOPMENT

I have hardly touched on curating interactive art in this thesis, but just as painting curators x-ray paintings to see how the layers of paint were built up, there will probably be a desire in the future to see how artists make interactive art today (there is a related issue of how to preserve interactive art, but that is already an active research field).

Some of my recommendations will assist in this process, such as integrating version management into the programming process, logging interactions and integrating design repositories into the programming. It is possible to envisage specific tools which would help analyse the programming process. For example, rather than colouring code by syntax, it could be coloured by age, and/or the amount of change or attention. Such a tool could conceivably be of use to programmers and artists, as well as to curators.

TOWARDS FUTURE TECHNOLOGY

In the *Cardiomorphologies v. 2* collaboration, I prototyped various aspects of an ideal system, and found indications that these aspects are useful in interactive art making. It is now time to develop technology that can be used more generally.

Khut's reluctance to use unreliable tools is, of course, an understandable response—being an interactive artwork on permanent display to the public, reliability is paramount. This brings to light a Catch-22 problem with developing new technology in the course of making interactive art—on the one hand, software will not be used until it is stable, yet on the other hand tools developed iteratively for a particular artwork must be used (and used realistically if it is to support creative work) in order to be developed and to become stable. A common response to this situation is to be conservative in the development of new software—to use only the tools that are available to make the art. This does not sit well with the innovative nature of creativity, so I recommend that future systems provide tools for more easily making reliable tools (or toys), rather than only tools which are focussed simply on making the end product. Khut alludes to this in the interview:

> *an important function for some of these toys is that they are "tools to think with" and that as an artist working with all these data, you need some way to keep it tidy and clear in your head about what's going on and what the relationship is between one thing and another? And [you need] ways of thinking about significant relationships and what kinds of things are significant in terms of behaviours of your data. And that's where something like "score view", [gt.score and] some of those analysis things, if there is way of providing and making all that information somehow more accessible and visible to you and so you can see that things make sense of that data, that's good. And these things don't relate directly with the actual art output but more to intermediary kinds of things. So in a way, sometimes when I work with programmers, there has been a strong element that they're actually helping you to make sense of what it is you want to do and I think that could be extended more into software tools.*

Specifically, there are two streams of research that could unite as they mature. The first is to make self-modifiable computing systems, as called for by circumstances such as George's inability to access the underlying algorithm of a Max C object:

> *If I took it apart, I wouldn't be able to put together again in a way that was apparent, so maybe if there was some way that in the documentation of that object or if you double clicked on it if there was a kind of help file, it just explained things and give you some equations for how to kind of explain what is happening to other people.*

and:

> *When I am working with other people, say like medical technologists, something like that, they want to see what you are doing. They will say, hang on, what this guy is doing or how did you arrive at this thing. Because I work between art forms and [with] doctors and stuff like that. It's about making some of those things transparent to the other people in the team.*

The second stream of research is to build tools for making toys easily. The gt.probe object is a prototypical example of this, made easy by Max's graphical nature, but constrained by Max's inability to introspect. There is a great deal more functionality and tailorability to be added and further usability studies to be undertaken with this approach. To take a specific example, Khut was retrospectively interested in the gt.score object:

> *I've been since thinking about how to work with sensor data, and thinking that what I need is some kind of way of printing out all the sensor data as a score through time like an orchestral thing with all your different parts layered over the top of each other. and now, I think, oh well, you could use score view in a way to do that but what it doesn't have is a system for labelling data so each of the different strands.*

The format and behaviour of such a 'score of data' is an interesting information visualisation problem in itself.

I further suggest engineering toy-making tools in a introspective system, if not for the new computing system outlined just above, then for more traditional object-oriented languages, such as Smalltalk, Java, C# or Ruby. In these systems, one could envisage a text editor that provides

the ability to highlight an object or section of code and have it parsed to determine its inputs and outputs and their data types, and an appropriate toy created at runtime.

Each of these research streams, a self-modifiable computing system, and tools for supporting toy-making, have long-term commercial applications in many creative fields, not just interactive art. Self-modifiable computing systems remove the rigid constraints around existing forms of computing that non-programmers are frustrated by every day. Self-modifiable computing systems would by definition be open source, but commercial opportunities to sell the system lie in providing added value, such as the prestige of creating the system, training and documentation, and consultancy. Tools to support computational toy-making also have commercial applications—particularly as tools for traditional developers working in interdisciplinary collaborations. Products embodying innovations in either research stream would help open the path to universal procedural literacy—where artists are able to develop their own vision without having to spend larger amounts of time either learning to program or work second hand through a programmer[1] , and where people from all walks of life are more empowered to use computers to explore and create.

1 Thank you to Tom Hewett for this elegant description

Appendix 1: Publications

This appendix is on the accompanying CD.

Appendix 2: Heuristic Evaluations of Max/MSP and Squeak Smalltalk

This appendix is on the accompanying CD.

Appendix 3: COSTART Coding

This appendix is on the accompanying CD.

Appendix 4: COSTART 2 Coding

This appendix is on the accompanying CD.

Appendix 5: Interview Questions

Questions for Artists

1. Background/History

    a. How long have you been making art?

    b. Why do you like doing it?

    c. How long have you been using computers?

    d. How long have you been making artwork using computers?

    e. What first made you interested in using computers for art?

    f. Why are computers important in art?

    g. How much do you think you understand the potential (technological) capabilities of computers?

    h. Do you program?

        i. What sort of programs are they?

        ii. Copied from books? Did you understand them? When do you think you reached the stage where you feel you could figure out how to understand any given program?

2. How many of your art projects have used computers?

    a. At what stage of your process do you think "hey, we could use a computer for this?"

    b. Could you give me a list, off the top of your head?

    c. Did you work with a programmer for all of these?

    d. Favourites or not favourites? Why?

    e. different models of collaboration: equal partnership, assistant, teacher, contracted consultant—which role do you prefer your technology collaborators to take? Why?

3. What's the hardest thing about working with programmers?

    a. Could you describe in your own words the process you go through with the programmer to create the art?

    b. How interested are you in the programming process?

    c. How involved do you become in the programming process? Is this enough, or too much? Can you think of anything that might make the right amount of involvement easier?

    d. Do you have a clear idea in your imagination of what you expect the finished work to look like? (if no:) How do you decide when it's "finished"? (experimentation) If you don't program, how do you experiment with the program? Does the programmer make special controls for you?

4. Do you think the programmer is able to influence the art or you as an artist?
   When? In what ways?b. What do you think about that?

Questions for Programmers

1. Background/History

    a. How long have you been using computers? What made you interested in them?

   b. When did you start to program?

      i. What sort of programs were they?

      ii. Copied from books? Did you understand them? When do you think you reached the stage where you feel you could figure out how to understand any given program?

   c. Why do you like programming?

   d. What languages are you most comfortable in?i. Why did you learn those?

2. How many art projects have you programmed?

   a. Could you give me a list, off the top of your head?

   b. Favourites or not favourites?

   c. What motivates?

   d. Candy/Mamykina paper, different models of collaboration—which role do you see yourself taking?

3. Collaborating with other programmers

   a. What kind?

   b. Did you stick to different tasks or both work on the same tasks?

      i. Did you use the same language?

The term 'Artists' in the following is too general. So choose one or several specific examples if you think they're useful.

4. What's the hardest thing about programming for artists?

   a. How do you choose tools?

   b. Broadly describe your development process?

   c. Do you show them what the problem is?

   d. Have you tried to show artists the code? Did they understand it?

      i. Have they ever changed it?

   e. Do you find that artists understand what you do? Are they interested?

   f. Do you add things to the program to make it easier for the artists to engage with the program?

   g. Is there anything you think that the tools could do to help support you communicating with artist?

   h. Do you use conventional software development processes to make art? (Like documentation, UML diagrams)

      i. Might you in future? When?

5. Do you think you are able to influence the art or artist as a programmer?

   a. When would you? In what ways?

   b. What does the artist think?

6. If working with artist who doesn't really understand the tools or the power general-purpose computing, how might you convey the idea?

   a. Demos? Diagrams

   b. Have you ever tried? Do you think it worked?

7. Open-ended discussion: When I learnt programming, I had a kind of object-oriented 'epiphany', where I got my head round the things you could do with OO that you could not with procedural, and it's kind of a mathematical, abstract way of looking at things. So some artists exploit that; some are kind of stuck in the 'if a, then b' mindset—it's hard to nail down though. Do you think it helps in art?

Appendix 6: Interview Transcripts

This appendix is on the accompanying CD.

Appendix 7: Interview Coding Table

Initial Coding Table

This table has memos attached, but they are omitted here for space reasons, and included in the final table at the end of this Appendix. Remaining (and renamed) codes from the COSTART coding are in green.

Technologist

	Interview Count
Suggesting	
do what's best	*
aesthetic/artistic vs. technical / pragmatic	***
easy/hard	
engagement level	*
active (steering) / passive (responding)	*
(passive-aggressive)	*
Predicting	
Revising Estimates	***
Work in != enjoyment out	*
Sharing meaning	
expressing problems	
specifically	*
metaphorically	***
ambiguity	*
question-asking	*
shows Artist the code (or at least the step-by-step operation of the code)	**
shows subset of the program made especially for artist	***
not sharing meaning/explanations failing	*****
artist makes technical decisions	**
Solving Problem	
preference for straightforward approach	*
familiarity	**
something that's 'never' been done before	*
finding previous instances	*
analysis vs. synthesis (analogous to experimentation / theorising)	**
set goal more satisfying	*
puzzle	*
pragmatic approach (how could we possibly do…)	***
compromise	*
iterative development	**
comparison with other creative activities	**
gathering rules from Artist	
by asking what counts as critical detail	*
by supposing scenarios (combinations of conditions)	*
identifying technology	
establishing communications channels before brain gets built	*
input harder to manifest than output	*
hard not to start 'brain' too soon	*
prototype in different media	****
providing finished product vs providing tools	*
Relating to Artist	
building with unforeseen expansion in mind	*
pressure from Artist	*
equal partnership	**
consultant / problem solver	***
contested hierarchy	**
tacit communication	*
receptive to suggestions vs likely to enquire	*
Teaching Artist	
shows Artist new possibility	**
influencing	
transferring skills/imparting knowledge	***
visual programming	**
answering questions	
using metaphor or disparaging language to keep from the nitty gritty	*
Motivation	
interest/curiosity/engagement	**
soulfulness/cool (enjoyment)	***
achieving goals/solution/completion (enjoyment/satisfaction)	******
excitement (enjoyment)	**
creative urge (enjoyment)	***
creative flow	*
learning skills (enjoyment)	*
research	*
control/power (enjoyment)	**

challenge (satisfaction)	****
building things	*
teaching (enjoyment)	*
elegance/beauty	*
Perceiving own role	
flexible worker	*
take himself out of the loop	*
Relating to Technology	
Building to be understandable	**
transparency	*
no need for external documentation	***
need for external documentation	*
Gulfs between development environments	*
Adjusting Parameters	
character of a system	*
Recording system behaviour (and annotating)	**
Opening up software to other control	*
Opening up software to transition, familiarity	*
preference for easy languages	*
preference for powerful languages	***
preference for relevant languages	**
preference for visual programming languages	*
preference for familiar languages	
long history	*
frequent	*
recent	*
taught	**
Relating to external people	
self-sufficiency	*
relying on other Technologists	*
other Technologists supply unexpected expertise	*
ability to get help influences tool choosing	*
Domain Knowledge	
insufficient	*
but necessary (eg maths)	*
specific technology	*
Relating to Art	
understanding art	
gratuitous synaesthesia' - relating many disparate elements - balancing act harder	
Relating to Structure	
Bottom-up vs Top-down	
Playing with small sections then building into structure/abstraction	*
making a sketch and discovering something new in the sketch	*
Concreting vs Abstracting	***
Building as you go vs building to design	*
object-orientation	*
from doing particular project	*
from experience	**
Learning Structure	
Seeing in action (max help)	**
Building communications networks	*
Important in some art	*
Artist Relating to Computing	
interested/seek understanding	****
not interested/outcome/audience only	**

Artist

	Interview Count
Ownership / Freedom	
Technologist has final say in certain things	*
Retaining Creativity vs delegating to Audience	*
Relating with Technologist	
valuing 'people skills' (intelligence, goodwill, sensitivity), devaluing grumpiness, arrogance	**
valuing agility/nimbleness vs devaluing inflexibility	***
valuing skill in desired environment	*
valuing education, expertise, devaluing 'dabblers'	****
valuing similar experience	*

valuing interaction design/brainstorming	*
valuing engagement	**
valuing speed of work/project management	**
valuing efficiency/'clean'	*
valuing cost	*
Technologist reaching past what they know (working with any material)	*
testing hunches (challenging T's experience)	*
Technologist going hermetic	*
craftsperson vs collaborator	***
media production methodology	**
art school' methodology	**
engineering methodology	*
respecting expertise	*
difficulty understanding problems	**
technologist seeking to know the project	*
Learning from Technologist	
learning Technology from Technologist's expertise	**
learning from collaboration	*
Sharing Meaning with Technologist	
developing shared meaning	*
Expressing Problem	
known rules mean solution	*
discovering unknown rules	*
playing with mapping	**
playing with parameters	***
communicating metaphorical meaning	**
desiring toolkit	
experimentation/playfulness	**
intuitive vs goal-led	***
expanding beyond original concept	*
Relating to Programming	
aware of difficulties	*
understanding structure	
specificity/concreteness	**
Breaking tools down into increments (production methodology related)	
[Agile Programming] straddling divide between holistic understanding of eventual delivery and addressing immediate needs	*
disinterest, vs engagement in concepts vs enagement in everything	**
front-end interest vs back-end interest	*
desire to program	*
Relating to Technology	
interaction design	*
being aware of potential	***
considering Technology as means to an end	*
show control	*
algorithm like a recipe	*
impact on the arts	**
eases labour-intensive art	*

Coding Table Restructuring Stage 1: Removing/Renaming categories.

Having added the interview data to the coding table the following categories from the COSTART coding were removed. All of them had no codes to augment them in the interview coding (my hypothesis for the reason is included below), or were covered elsewhere in the category scheme:

TECHNOLOGISTS' CATEGORIES

Suggesting->**Like/dislike**, removed because it is too vague (more specific reasons for technologist actions are covered elsewhere)

Suggesting->**Commitment level**, removed because it didn't arise in the interviews, perhaps because the collaborators were experienced, so their commitment was taken as read.

Sharing meaning with artist was renamed "Sharing meaning".

Rules for problem expression was renamed expressing problems, since topics other than 'rules' were covered.

Communicating with Artist and its subcategories were removed; they described personal qualities that are either covered elsewhere (e.g. in relating with artist), or would be filtered out by having the artist and technologist choose who they collaborate with.

Solving Problem->**fiddling/play vs. problem-solving** was incorporated into "analysis vs. synthesis". However no codes were found except "set goal more satisfying". Also, in the 'motivation' section technologists express satisfaction at solving problems.

Creating Structure has become "relating with structure"

Known Rules Mean Solution was removed because it is somewhat contradicted in the interviews. In particular, [IA2] talks about "dynamic sets of information that had probabilities and tendencies in them rather than prescribed discourses". Although the computational form of the work being discussed could be expressed in terms of 'rules', any 'solution' to the problem represented by the art work relies on human interpretation of the "probabilities and tendencies". The category originally arose from an artwork in which human interpretation was perhaps not so important in terms of the 'correct' functioning of the computer system.

Computer needs rules, **Difficult to express rules to the computer**, **requiring logic**, **requiring specificity**, **requiring sequentiality** were removed because no-one addressed these points in their responses, perhaps because it had become tacit in the expe-

rience of the respondents. In the case of the latter three categories, I suspect that the experience of the respondents had given them skills to gather rules from artists without these requirements.

Describing in terms of technology has been superseded by 'answering questions'.

Suggesting->**Easy/Hard** was removed as it wasn't mentioned except in the context of being motivated by a challenge.

Relating to artist->**Requesting technological understanding** was moved to "artist relating to computing->interested/seek understanding".

Relating to artist->**Wondering aloud whether artist wants to learn code** was removed, as it was an isolated occurrence.

Relating to Artist->**providing finished product** was renamed "providing finished product vs. providing tools"

Teaching Artist->**Show new stuff** was renamed "shows artist new possibility"

Teaching Artist->**Inspiring** was removed as no respondents mentioned it separately from other forms of teaching.

Teaching Artist->**Test Reactions** was removed as no respondents mentioned it separately from other forms of teaching.

Motivation->**Money** was removed as no respondents mentioned it. This originally arose as a hypothetical comment in a technologist meeting (and as remuneration for programming art is generally fairly low on the technologist's pay scale, the hypothesis is unlikely to be grounded in reality).

Motivation->**Ownership** was removed as no respondents mentioned it.

Perceiving Own Role->**support team** was removed as no respondents mentioned it, except in the context of relating to the artist as a consultant (which is dealt with in that category).

Perceiving Own Role->**Professional** was removed as no respondents mentioned it. (One artist talks about "proper" computer scientists, which is dealt with in the artists' categories.)

Relating to Technology->**Computers' virtue...** was removed as no respondents mentioned it.

Perceiving Own Role->**Program core is like an engine** was removed as no respondents mentioned it, it being an isolated incident. However, another technologist mentions the program core as being a 'brain' linking sections of the body, so this metaphorical way of describing computers is more prevalent than isolated incidents. This is related to the category "using metaphor or disparaging language to keep from the nitty gritty".

Domain Knowledge->**Real-world conditions** was removed, because it refers to a one-off incident (where a VR designer used approximations to real lighting to attempt to create an un-real lighting effect—a limitation of a particular application).

Relating to Art->**A priori/A posteriori interpretation** was removed, since it is superseded by the relating to artist and suggesting sections.

Relating to others is renamed "Relating to external people" for clarity

ARTISTS' CATEGORIES

Responding To Suggestions was removed as it is covered more carefully in the other categories, particularly "relating with technologist" and "Communicating with Technologist".

Relating With Technologist->**Perceiving complexity in technologist's action** was removed since no respondents mentioned it (except through relating with technology). It is possibly tacit as a result of experience.

Working Apart from Technologist, Relating With Technologist->**Reluctant To Delegate**, **Delegating 'interaction coding'**, **Being Used to hands-on approach**, and **Watching Technologist work** were all removed because the interviews showed that different artists are differently content to delegate (see particularly "craftsperson vs. collaborator" and "'media production' and 'art school' methodologies").

Feeling Pedantic was removed since no respondents mentioned it.

Wanting to learn to code was removed, since no respondents wanted to learn to code. According to both technologists and artists, artists vary in their desire to learn to code, and the level to which they're interested (see "relating to technology")

Communicating with technologist->Trust was removed for the same reason as it was removed from the technologists' section. Other codes in the "communicating with technologist" category can be incorporated into "relating with technologist"

Sharing Meaning with Technologist->**Rules for problem expression** has been superseded by "Expressing Problem"

Sharing Meaning with Technologist->**trying to adjust thinking to work more closely with programming** was removed since no respondents mentioned it.

Sharing Meaning with Technologist->**Asking Questions** was removed as no respondents mentioned it. The question-asking to solve problems seems to be mostly by technologists of artists.

Expressing Problem->discovering unknown rules->**in response to unanticipated combinations of conditions** was removed since no respondents mentioned it.

Expressing Problem->**Using vague language** has been renamed "communicating metaphorical meaning"

Expressing Problem->**Desiring toolkit** was removed since it wasn't mentioned by the artists except as direct responses to my questions (so the risk is that I put words into their mouths). However, the topic is dealt with in the analysis through the theme of play.

Expressing problem->**Experimentation** has been renamed "experimentation/ playfulness"

Expressing Problem->**wanting to define limits within which the computer can be random** was removed since no respondents mentioned it. It is however related to the point above about the computer needing rules.

Expressing Problem->**requiring logic, specificity** and **sequentiality** were removed for the same reasons as they were removed from the technologists' section.

Expressing Problem->**admiring sequentiality** was removed since no respondents mentioned it. The nearest point was from [IA1] who "respected [technologists'] area of expertise".

Expressing Problem->**describing in terms of effect** was removed since no respondents mentioned it, beyond "communicating metaphorical meaning" and being "interested in the front-end vs. back-end" of the art systems.

Compromising Quality was removed since no (artist) respondents mentioned it (non-quality compromises are mentioned by the technologists).

Relating to programming->**Constrains Ideas** is superseded by "Relating to technology->being aware of potential"

Relating to programming->**Programmer Flair** is superseded by the "Relating with Technologist" categories. However, there was no further mention of 'programmer flair'—perhaps it becomes tacit with experience.

Relating to programming->**Understanding Structure** was removed because no (artist) respondents mentioned it. The nearest was [IA1] discussing using playful interfaces to understand an algorithm's place within the structure.

Relating to programming->**too specific** was renamed "specificity/concreteness", as no explicit opinions were given on to what extent programming was 'too' specific (and for whom).

Relating to programming->**according to specifications** was removed because no (artist) respondents mentioned it, preferring instead to discuss 'strong concepts' and metaphors, each dealt with elsewhere.

Relating to programming->**layers** was removed because no (artist) respondents mentioned it. However, there are implicit layers in the front-end/back-end discussion, and so the concept of a continuum of layers (through which an artist can dig deeper/see the big picture) has emerged as a subtext to the theory, and is dealt with in the analysis.

Relating to programming->**world of mystery** has been superseded by "being aware of potential"

Relating to technology->**need full control over system** was removed because no (artist) respondents mentioned it, except in the context of 'art school' methodologies.

Relating to programming->**rigidness** was removed because no (artist) respondents mentioned it. The concept is covered amply in Study 1.

Relating to programming->**inspiration by technology**, and **creates aesthetic** were removed. In part they were covered in "impact on the arts", but perhaps these things were also not mentioned because the artists, or the collaboration as a whole, are experienced enough to be able to subvert technological clichés (which arise from rigidness, as described in Study 1) and to 'resist' converting technological limitations into features of the work.

Coding Table Restructuring Stage 1: Rearranging the hierarchy—the emergence of 'Attuning'

In order to make a coherently structured grounded theory, it was necessary to attempt to collect the categories into super-categories, and from there into super-super-categories, and so on, until a small number of parent categories emerged. This section describes that process.

1. "Relating with" categories are renamed "relating to" for consistency, and to emphasise the subjectivity of the categories.

Hitherto, the artist and technologist categories had not necessarily been identical. I wanted to arrange the categories to bring out any reflexive/mirrored categories, to better explain the relationship between artist and technologist from both angles.

2. "Expressing problem" and "Solving problem" are both renamed "relating to problem", for consistency with other categories, and between artist and technologist.

3. Many existing super-categories can be grouped under "Relating to Project" (predicting, ownership/freedom, motivation, suggesting, perceiving own role), "Relating to Artist/Technologist" (teaching/learning, sharing meaning), "Relating to Problem" (domain knowledge) and "Relating To Technology" (relating to programming, structure).

4. All of the categories fit into one of those super-super categories, except the technologists' category "Relating to external people". Since this category contains nothing particularly saturated or interesting, I will omit it from the theory.

5. So the remaining super-super-categories are shared and mirrored between artist and technologist, which implies that the commonalities and differences—the give and take—between artist and technologist must be negotiated as part of building the shared understanding necessary to a successful collaboration. In my search for a concept to unite these super-super-categories under a single concept representing this give-and-take, I was helped by one of the artists describing the same thing:

> *"And then its just a question of cross checking with everyone and then having other people come back with their interpretation and its signal, signal, signal, transmission, signal until you all start to think ok we've attuned ourselves somehow."*
>
> [IA2]

The concept of *attuning* (and its evocations of orchestra tuning and car engine tuning) seems to sum up the super-super-categories well, and will do as the unifying concept for the grounded theory.

Here is the final coding table:

The Final Coding Table

Category	Interview Grounding Count		Memos
Attuning			
Technologist			
Relating to Project			
Motivation			Distinguish Enjoyment and satisfaction as per [IT4]. Satisfaction requires the final form
interest/curiosity/engagement	**	[IT4]	art - [IT4]'s paragraph describing it all. Variety of reasons.
soulfulness/cool (enjoyment)	***	[IT1] - "cool", [IT4] "sense of expression beyond simply […] processing" [IT3] [Web Design] soulless	Future: No reason why computing shouldn't be taught at art schools, not just engineering schools
achieving goals/solution/completion (enjoyment/satisfaction)	******	[IT1] "getting something done" - annoying that it's not finished. [IT2]. [IT4] "least favourite no sense of completion" - "conversely move from exploration to fruition", [IT3] "problem solving with a defined end" + "if someone sets me out a goal that I find hard to nail down ... [it's] going to make me unsatisfied"	prog. Because of control etc that programmers like to have a sense of achievement/completion. So artists: express artwork in completable terms, [IT3] compares with philosohy where you can't just plug away and get to a position"
excitement (enjoyment)	**	[IT2], [IT3]	art
creative urge (enjoyment)	***	[IT1] "interesting ideas", [IT2] "experience that I help create", [IT4] "imagination and design"	art. Computing is *"beyond* imagination and design"
creative flow	*	[IT1]	prog
learning skills (enjoyment)	*	[IT1]	prog
research	*	[IT1]	art coll
control/power (enjoyment)	**	[IT2], [IT4] "lack of resource limitations"	prog. Servitude
challenge (satisfaction)	****	[IT2], [IT3] "did poorly… because there wasn't a challenge" + "found [web design] repetitive and unchallenging" + "I was more excited about doing the things I didn't know how to do"	prog + art - attracted by the inability to do something
building things	*	[IT4]	prog
teaching (enjoyment)	*	[IT4]	coll.
elegance/beauty	*	[IT4]	prog
Perceiving own role			
flexible worker	*	[IT3] "they were looking for [someone] to be able to 10 or 15 different things"	
take himself out of the loop	*	[IT4] as a result of the gulf of interpretation, "Give them control" "critically… it makes it really clear what the dynamics of the system are as opposed to what my interpretation is as well"	toys
Predicting			

Revising Estimates	***	[IT4] "Have to make clear and explicit" + "make sure it's actually as important as it is difficult" [IT3] "to get the shell right takes more time than you expect and to get the brains and the decision making right takes a lot less time than you expect" - car comparison	the new part takes less than you think, the 'boring' get everything working and talking takes more time.
Work in != enjoyment out	*	[IT3] - preference for straightforward manifestations of good ideas	
Relating to Artist			
building with unforeseen expansion in mind	*	[IT4] "get a good sense of which parts they're interested in, making [them] extendable or expandable"	
pressure from Artist	*	[IT1] - "he kept standing there"	"needless" pressure? Time pressure OK, personal pressure less so? Or maybe combination.
equal partnership	**	[IT1], [IT4] (see below)	Equal partnership -> less likely to make tools?
consultant / problem solver	***	[IT2], [IT4], [IT3] "if they want you to do something and you tell them that it's not going to work and it's your job to know, then they'll believe you and they'll ask you what will work"	Key to successful relationship
contested hierarchy	**	[IT4] x 2 - planning; unspoken boundaries	
tacit communication	*	[IT4] ("implicit and unspoken aspects"…"can become entrenched without being specifically discussed"…"I really try and contest these boundaries and limitations")	doesn't an artist mention that these are made clear? See ambiguity
receptive to suggestions vs likely to enquire	*	[IT1]	Artists: enquire!
Teaching Artist			
shows Artist new possibility	**	[IT1] - from Max help! Nice quote. [IT2], "listen first, then offer what I could" using various form of communication.	Artist is inspired by examples to get an idea of possibilities (danger is that just pick nice ones - cliché).
influencing			
transferring skills/imparting knowledge	***	[IT4], [IT3] "explaining to someone what I'm doing is a really simple way of unerstanding for myself" + "it becomes on the heads of people like myself and yourself to help [artists] get educated in the same way they help educate us in the making of art"	see also below re artist engagement with tech
visual programming	**	[IT4] "it's a domain where artists feel confident enough or have an interest in becoming conversant" + patching environments"it's possible to use the environments and understand the visual syntax without really having a good sense of programming or structure"	VPLEs inspire confidence because they look familiar (like GUIs, physical arrangements). Does this code belong here? [IT4]: "physicality" Conversant is a nice word because it implies PL feedback (attuning!). Nardi
answering questions			
using metaphor or disparaging language to keep from the nitty gritty	*	[IT3] "will always answer questions" "incrementally more information"	No evidence except from [IT4]/Max of teaching programming. [IT3] gives

			incrementally more info - perhaps this is why technologists are good - they do the 'dig deeper' thing that PLs should
Suggesting			
do what's best	*	[IT1] - but what's best? (given constraints)	
aesthetic/artistic vs. technical / pragmatic	***	[IT1] both sides "pragmatic", "he didn't let me get away with it"	
engagement level	*	[IT4] "bring as much as I can aesthetically and creatively to a project"	
active (steering) / passive (responding)	*	[IT1] - steering	
(passive-aggressive)	*	[IT4] making what was asked for, even though there were other possibilities	But the toy was a way for artists to see for themselves the (structural) limitations of an approach -- or place within the system. Add this to reasons for toys. Passive-aggressive approach working well here. Note good and bad p-a approcaches - to do with the motivation for the suggestion? Toys are good for taking the programmer out of the loop
Sharing meaning			
expressing problems			
specifically	*	[IT2] - "good idea of the features we needed" - but they changed	Theory: specific features gives programmer easy life, therefore better able to create a tool. A: if not sure of specific requirement, be sure of general requirements and commission a tool.
metaphorically	***	[IT2] - "my best explanation without boring the artist" and "artists that have a high-level idea" so conversing in terms of the artist's experience. [IT4] "we want the [piece] to have this sort of feeling" "big gulf of interpretation"	
ambiguity	*	[IT4] 'do you mean this, this and this?" "take a best guess .. With the knowledge that.. There may be backtracking involved"	toys reduce ambiguity
question-asking	*	[IT3] "lots and lots and lots of questions"	see above too. [IT1] and [IT2] might mention this too…
shows Artist the code (or at least the step-by-step operation of the code)	**	[IT1] - Max, [IT2], PD, but otherwise not.	
shows subset of the program made especially for artist	***	[IT1] (hiding code), [IT2] whole app "content creation", [IT4] "exposes the important parts of that development work to the artist as early as possible" (resolves ambiguous/difficult design decisions)	Design rec: control panels. Speccing a tool for the artist is easier than speccing an audience experience. Interface builder makes 'quick' construction of control panels and new controls. However, widgets insufficient for artist to use on own.
not sharing meaning/explanations failing	*****	[IT1] - "some of them don't necessarily know enough", "I was never sure that [artist name] got what I was trying to do" - "perhaps he didn't really want to [experiment]". [IT4] "no one involved in the project had a good understanding of the sort of algorithm that	Artist didn't understand that a semi solution wasn't the solution (like Artist's 21^22). [IT4] saw a need to explore 'the sort of dynamics it could produce, the extreme points, what you could actually get this algorithm to do". Exposes

		had been chosen and discussed"	potentials
artist makes technical decisions	**	[IT4] "artists try to take into account things they perceive as being technical interests without necessarily making those explicit" + "choice of the algorithm and some other technical aspects were specified in the process of designing the work which I don't think is necessarily appropriate particularly when there wasn't a great deal of knowledge or experience with these technical artefacts"	Relate to 21^22 again.
View of Artist Relating to Computing			see also vpl stuff
interested/seek understanding	****	[IT4], [IT3] "80% of the time the artist wants to be involved in everything … not programming.. But… how the decisions I'm making are going to affect the outcome" [IT3] "by better understanding what I'm doing they can better design their own works and better understand the constraints on them" [IT3] "artist practice is always better from them engaging in the[ir] technologies ... the more they learn, the better art practitioners they'll be"	see alse teaching/skill exchange. 2 kinds of artists - those who care about the back-end (and want to program) and those who don't, and are creating the experience
not interested/outcome/audience only	**	[IT3] "other sort of people who have a vision and simply want someone to manifest that vision" [IT3] "at the same time there is some benefit … from people having no idea about the technology and just asking for things that no one has thought of asking before because [everyone else has] a whole bunch of preconceptions about how to use technology.	So does this duality correspond with the strength of the artist's vision at the outset? See consultant/support. Re IT3's 2nd point: does having no idea really result in new questions? Surely the 1st questions are going to be the ones that *everyone* has asked before?
Relating to Problem			
preference for straightforward approach	*	[IT3] "audacious … fell slightly short of [their] lofty goals"	Technologist prefers straightforward solid ideas, but this is probably not universal.
familiarity	**	[IT1], prefers + shows artist examples	
something that's 'never' been done before	*	[IT3] - "putting together things that have never been put together before"	
finding previous instances	*	[IT3] "its only familiarity that gives you that skill, not a technique or whatever"	Techs watch other techs a lot to see how they did it (open source good)
analysis vs. synthesis (analogous to experimentation / theorising)	**	[IT1] synth, [IT4] synthesis	[IT4] "synthesis of ideas, between design, intention and expression"
set goal more satisfying	*	[IT1]	see motivation goals
puzzle	*	[IT1]	
pragmatic approach (how could we possibly do…)	***	[IT1]x2, [IT3] "you could choose from a hundred different solutions to most problems"	individual preferences dictate choice of solution? Compare with insufficient domain knowledge
compromise	*	[IT3] - "something gets in the way, like they don't have enough money, or there's not enough time or space… what usually happens is you say OK we can do that but only at the	see [IT4] important as it is difficult - compromise

		sacrifice of something else"	
iterative development	**	[IT4] "in situ process", [IT4] "exposes the important parts of that development work"	I Wonder if agile development is close to art development methodoloies? Important/difficult parts -> iterative; unimportant/straightforward parts -> noniterative. See also [IT1] pressure from artist
comparison with other creative activities	**	[IT1] writing, designing	motivation
gathering rules from Artist			
by asking what counts as critical detail	*	[IT4] did some of this… see resolving ambiguity	
by supposing scenarios (combinations of conditions)	*	[IT3] "I start drawing scenarios" when it's something he's never seen before. Some of these would be non-representational actions	Testing edges of system
identifying technology			
establishing communications channels before brain gets built	*	[IT3] "I want everything to talk to everything else before I start working on how decisions are made"	if there's no input or no output then it's hard to simulate accurately what's going to go in the middle… so design tools to support capture of input (see also annotation).
input harder to manifest than output	*	[IT3] "usually input is the hardest thing to manifest; output you can fake pretty quickly"	Prob. Because input has the human element
hard not to start 'brain' too soon	*	[IT3] "the hardest thing about programming… is curbing my urge to just start straight away"- more precisely "it's dealing with the operating system or… protocols or… vagaries. To get the shell right takes much more time than you expect"	Even programmers need assistance with understanding computing (the 'boring' bits)
prototype in different media	****	[IT1] - sketching (metaphor), [IT4] Python-> C, [IT4] - initial questions, then work out on a bit of paper, [IT3]: start by drawing scenarios even just on squares of paper	
providing finished product vs providing tools	*	[IT2] - "possibie behaviours"	See also goal motivations
Domain Knowledge			see also familiarity
insufficient	*	[IT1] "there's nobody saying this is the answer"; can't know all the possible ways	see also attracted to challenge
but necessary (eg maths)	*	[IT1] accelerometer	
specific technology	*	[IT1]	
Relating to Technology			
Building to be understandable	**	[IT4] "patches that are clearly understandable", [IT3] "avoid having to explain to people how to use it"	mentioned in costart
transparency	*	[IT3] "what would be beautiful would be if I could write software that wos so transparent that the people using it could see inside it without my help."	See technologist providing 'dig deeper' service. Argument for 'dig deeper'
no need for external documentation	***	[IT1] - "Max is kind of a little bit like that". [IT4] - "the need for system construction or specifications kind of idea have never really	VPLs as replacement for ER diagrams - agile programming (code is the documentation). Tools imply production

		happened" + "not at all, except … some tools imply various things about processes in the way they work [e.g. svn]"	processes (because they are inflexible)
need for external documentation	*	[IT3] "big projects … have to have a central repository of decisions"	design - support for this in the code?
Gulfs between development environments	*	[IT4] "gaps and gulfs ... that I feel very keenly."	Look at his pg for design ideas
Adjusting Parameters			
character of a system	*	[IT4] "comes from that sort of control"	produces meaning both technically and aesthetically
Recording system behaviour (and annotating)	**	[IT1], [IT4] (and annotating)	Design Rec: recording + annotating particularly input
Opening up software to other control	*	[IT4] (inter-program control)	What does this mean? 1) communication between programs, 2) communication between programming tools (version control knows what else is happening, etc)
Opening up software to transition, familiarity	*	[IT3] "mythical toolbox" "everyone's using different programs and different ideas and have different backgrounds and even in different physical and spoken languages like you often end up doing things that a hundred other people have already done before and that could and should be avoided"	Design rec: AI assistance for design patterns. See also getting help
preference for easy languages	*	[IT1]	Design rec: intelligent help/code search
preference for powerful languages	***	[IT2], [IT4] when performance necessary, [IT3] object-oriented	speed S
preference for relevant languages	**	[IT2] "ability to provide the features we needed to offer the artist", [IT4] "depends on… the kind of relationship I have with the artist"	
preference for visual programming languages	*	[IT4] - quick to develop, transferred between platforms, abstractions are fairly clean. Many people are conversant with those environments more so than other languages, excellent permeabilities	Interesting words conversant and permeabilities. Perhaps interactive languages are more like a conversation. Synaesthesia s
preference for familiar languages			
long history	*	[IT4] "the last five or six years"	
frequent	*	[IT2]	
recent	*	[IT2]	
taught	**	[IT1] [IT2]	Strongest preference expressed! No other history commonalities
Relating to Structure			see also object orientation
Bottom-up vs Top-down	*	[IT4]	Theory: very top-level design is implemented bottom-up, because the patch systems (at least!) encourage it. Yet most programming design/macro thinking is a step up from that. How to manage that step up? Max patch wrapping, only better. That Media Lab

			thing. Bow tie diagram with artistic and computational layers of thinking, Bit in the middle is very small and encapsulates a lot of meaning! Idea is for the artist to pust at the far boundary, into greater computing knowledge.
Playing with small sections then building into structure/abstraction	*	[IT4] sees how small parts operate and has to think about how to fit them into a system… "top-down is very different from the bottom-up approch that most patching environments encourage … it's quite hard to switch back and forth between those things"	
making a sketch and discovering something new in the sketch	*	[IT1]	
Concreting vs Abstracting	***	[IT2] - relates in terms of artist's experience rather than abstract programming structures. [IT4] "you want to have some degree of separation between the system and its particular configuration at any time" [IT4]: "realtime response is more about control than program"	[IT4] says even in Max there is a difference between "configuring the system and modifying in different ways. I think it's important to preserve that." At first it sounds like anti single, universal programming, but it's OK, these things lie on a continuum, and it's OK to make a difference so long as the difference can be changed. [IT4] 2: Realtime environments/VPLs may make it more difficult to learn "the grand beauty" because it's too immediate and physical!
Building as you go vs building to design	*	[IT3] "there is a benefit in small scale things in the fluidity that comes between the collaborators [miscommunicating]" etc	see iterative, ambiguity, no formal documentation
object-orientation	*	[IT3] "everything complicated inherited from something simple" arm, hand, finger	OO is a good question from study 1. 2 streams of thought - one that OO isn't crucial for a lot of interactive art (influence of Max makes OO irrelevant because it's half built-in), other that it is great and should be used for everything.
from doing particular project	*	[IT3] "that was the only simple change I needed to make instead of having to change every single piece. And so that kind of turned me from not quite understanding what object oriented programming meant to wanting to only use that, because then I knew, it's definitely the way to go"	
from experience	**	[IT1], [IT3] "programming should be taught in high school" "most people don't know what's going on inside a computer so that when it crashes they don't know why … it's not that they're deficient it's just that they are naive"	See [IT4] below. Computers should come with a computing course (find out why this computer crashed). maths learning.
Learning Structure			
Seeing in action (max help)	**	[IT1], [IT4] "seeing effective examples of good things with and without explicit pedagogical instructions"	The only way [IT4] has learnt!

Building communications networks	*	[IT2] (APIs)	
Important in some art	*	[IT2] (generative)	

Artist

	Interview Count	Grounding	Memos
Relating to Project			
Ownership / Freedom			
Technologist has final say in certain things	*	[IA1]	Tie in with [IT4]'s contested hierarchy
Retaining Creativity vs delegating to Audience	*	[IA1] - "the key idea which separates an interactive work from a non-interactive work is how much the artist wants to be the author and what kind of experience they want to give the punter"	So there's computational importance vs. servitude (elsewhere), and artist control vs audince control. Does artist control relate to computational importance? Is servitude used to help the audience manage their experience? Future work.
Relating to Technologist			
valuing 'people skills' (intelligence, goodwill, sensitivity), devaluing grumpiness, arrogance	**	[IA1] values "how that person will work with the rest of the team", + below	see also "speed of work"
valuing agility/nimbleness vs devaluing inflexibility	***	[IA1] "he turned out to be very inflexible and grumpy" "wouldn't meet us part way and that was turned into a bit of a nightmare" [IA2] on dynamic collaboration + "In a programmer I look for linguistic and conceptual nimbleness and agility"	Nimbleness of collaboration skills as well as conceptual nimbleness. Agile programming too.
valuing skill in desired environment	*	[IA1] "choosing a programmer is based on the environment I think it's going to be developed in and the advice I get from the programmer about the appropriate environment."	Interesting - if programmer suggests environment, why is choice of environment used to choose programmer? Indicative of a pre-selected preference. Could be a case of artist making technical decisions.
valuing education, expertise, devaluing 'dabblers'	****	[IA1] computer science and engineering and who can program C a proper programmer + "I chose someone whe was essentially an arts student dabbling … who wasn't a real programmer so really that's why I've got quite firm about [quality of programmers]" [IA2] expertise + "broad range of intellectual and personal experience"	
valuing similar experience	*	[IA1] "if they'd done similar work before"	[IT3] exploits this
valuing interaction design/brainstorming	*	[IA1] (brainstorming seems to be collaborative)	and other related skills? Even in large groups! Perhaps IA1 is talking about engagement (see below)
valuing engagement	**	[IA1] "it's very difficult to lock it off and send it to the programmer and have the programmer say you've got it wrong" and accompanying paragraph [IA2] "energy and enjoyment … they are the driving force for a while"	
valuing speed of work/project management	**	[IA1] "[he] had been very fast" + "what makes a good programmer is someone who	in a commercial job?, but this was why he was chosen for new project. Future:

		can understand complex ideas quickly" - this is also sensitivity	do artists choose programmers for personal work differently than for commercial work?
valuing efficiency/'clean'	*	[IA1] "clean" - ambiguous meaning	could be a comment on the programmer's visual aesthetics, or mode of working in a collaboration
valuing cost	*	[IA1] "reasonably priced"	for a commercial job
Technologist reaching past what they know (working with any material)	*	[IA2] "not threatened by reaching out past what they know about"	compare with [IA1] valuing similar experience; [IT3] and programmers attracted to challenge
testing hunches (challenging T's experience)	*	[IA2] "it's a process of testing hunches with a bunch of people some of whom have technical abilities"	see also [IT3]/[IA2]/[IA1] naïve disinterest in programming
Technologist going hermetic	*	[IA2] "part of being a good programmer is often the ability to go into a somewhat hermetic world and enjoy setting up that world and finding focus in it"	I love 'hermetic'! From open world to closed system. Relate with technologists control/power. [IA2] values both - again holistic. Holistic is straddling frontend-backend, big-picture/dig deeper, people-technology and gentle-deep programming! Technologist attunes the sides of these continua. Need a pg summarising the types of attuning (art-tech, front-back, gentle-deep)
craftsperson vs collaborator	***	[IA1] "I've hired them… I they're constructed as a bit more as a craftsperson on the project … I've not actually done a collaboration with a programmer, which would be really interesting" [IA2] "I much more enjoy a kind of consultative, collaborative process" + "I wouldn't prioritise one over the other, but I do find that it is a dynamic situation ... [everyone] is the driving force for a while"	I've only interviewed artists who hired programmers - Draw out [IA2]'s last point (long quote in interview) - collaboration is a dynamic situation - not a contested hierarchy, and not as Mamykina and Candy have it, and everyone needs to be nimble enough to allow that to happen.
media production methodology	**	[IA1] - "I'd already been making film and video for 20 years … so what I did is set about to find a production team in a similar way" + "[in film] it's like if you wanted to work alone what's wrong with you, you individualistic romantic shit. So I've always worked most comfortably with teams"	IA1 wished she'd approached that phase differently, but she learnt the software which were most analogous to the environments in film and video.
art school methodology	**	[IA1] "there is definitely a school of artists who have an attitude if you don't do it all yourself it is not art" + "I know artist friends … who have tried to learn from [programmers]. I think that's great as long as that's an agreement and it's a respectful exchange"	see learning technology. Must it be an exchange?
engineering methodology	*	[IA2] anti: "I enjoy less, and I find less productive a kind of engineering model of working where a brief gets written and instructed down the line to the programmer ..	Compare 'satisfactory' with programmers' satisfaction. Design rec: for managers and creative environments - support agile development

		occasionally because of the institutional farmeworks that I've found myself in I've been compelled to work that way ... I find that it meets certain pre-determined schediles but it delivers less than satisfactory results"	
respecting expertise	*	[IA1] "it's a very respectful position … I think it's a an equal thing"	mutual respect [previously in code: (related to flexibility - will only respect exp. If flexible)???]
difficulty understanding problems	**	[IA1] "the hardest thing [about working with programmers] is that there is a point at which I can't help with the problem or engage with the problems." + "[for] a programmer, the problem for them is across the whole scope"	this could be in problem expressing…? OTOH, artists make tech decisions, OTOH tech asks them about things that 'don't affect the front end'. Future - how do artists feel about techs making aesthetic decisions (the problem for them is across the whole scope - [IT4] brings everything he can to a project). Can techs help making aesthetic decisions? How does this compare with techs feeling about artist's making tech decisions? See also holistic understanding (and people skills)
technologist seeking to know the project	*	[IA2] "the means by which the programmer is both creating that knowledge and acting on that knowledge formed thus far is to specify the suite of needs that the program might have to address in the next stage and therefore through a quite rapid and in some ways messy iterative, iterative, iterative process you watch the project emerge in a way"	Relates to question-asking, prototyping on prog side
Learning from Technologist			
learning Technology from Technologist's expertise	**	[IA1] "I've learnt generally a lot about computational systems and generative systems [etc]." "I've learnt a hell of a lot from programmers but I'm not looking at them to teach me code" + "I wonder if [artists who want to program are] really respecting that that person has done 4 years of comp sci ... it's a whole way of thinking, and a whole way of conceptualising"	A propos of "dig deeper". See my comments about teaching programming in art school. [IA1] may think it can't/shouldn't be done
learning from collaboration	*	[IA1] "I expect that the programmer and I in a good professional environment will both teach each other different things"	
Sharing Meaning with Technologist			
developing shared meaning	*	[IA2] "the hardest thing about working in any multicultural situation … is finding some sort of communication system where everyone has at least 70% chance of referring to the same thing … and maintaining a good shared memory of that language and of decisions	[IT3] spoke of central repository. Toys make physical/computational shared language. Generates meaning for all the stakeholders simultaneously.

		made"	
Relating to Problem			
known rules mean solution	*	[IA2] anti: "dynamic sets of information that had probabilities and tendencies in them rather than prescribed discourses"	[IA2]'s work is all about not laying out the rules. So there are meta-rules. This parallels specification recommendations opposite.
discovering unknown rules	*	[IA2] "the settings form a kind of log of performance and requirement that the programmer can go to"	Rules make sense to artist. Dealt with less by technologists (originally technologist who was also artist conception). Relate to [IT3]/[IT4] wish for logging of I/o. Recording of IO to toys. Max has separate systems for recording configurations (presets) and data streams (seq or whatever it is)
playing with mapping	**	[IA1] "It's been great to do a couple of programs in Max and to play with data mapping and develop the rules" + "I've very much enjoyed with [Art title] I've developed the rules for the world as to what content can get rolled out based on metadata enquiries"	"underst[anding] the language of the algorithm" (by playing with it). See also being aware of potential.
playing with parameters	***	[IA1] "that gave us a tool to say well we can choose certain parameters" + "Instead of [programmer] just doing and you saying can it be more squiggly … I found I understood the language of the … algorithm just by playing with the parameters and understading … how they had broken down this organic thing into a series of algorithmic..." [IA2] "often the artworks that are being made are manipulable systems of some kind. I keep coming back to this metaphor, they're things that need to be **tuned** in particular ways and often the **tuning** is best achieved from my end of things if I've got a kind of array of sliders by which I can adjust things." + "everyone who has responsibilities and curiosities about the project has some way to gauge impact and requirement ... you can mess with it and move the sliders around and say see how the world's altering from [your] point of view."	Toys discover rules. Toys allow artist to choose relevant parameters. Toys develop character (last 2 things are 2 sides of same coin). Toys help attuning. Instrument tuning, Piano tuning (strings in phase), engine tuning and now computer tuning!
communicating metaphorical meaning	**	[IA1] "it's about helping them getting a handle on the aims of the work, the kind of effect you want it to have on the audience … broadly speaking" [IA2] "in myself and communicate to the others a metaphorical understanding of the objective … I shy away from a detailed description of the project, but I try to offer again and again quite carefully considered metaphorical accounts of the project, it's like this, it's like this, it's like this" ... "signal,	seems to be a 'tell the programmer' perception. Not interested in the programmers' views. [IA2] - communicating metaphor is the 1st stage in XP. Multiple metaphors might be the artistic version of 'dig deeper'. Perhapse leaves room for technologist's aesthetic interpretation. The deep levels get attuned and so do the shallow levels. Relate [IA2]'s repetition to iterative

		signal, signal, transmission, signal until you all start to think ok we've **attuned** ourselves somehow."	question-asking of the programmer
experimentation/playfulness	**	[IA1] "knowing how to insert [playfulness] into a structured system is quite hard" + "[technical producer was helpful] because it would have been hard for me and [artist] to get [programmer] to make that [playful interface], that quickly"	[IA1] says agile programming does it. Design rec. need to build support for play into computing's structured systems (programming languages). Was it Tim's idea to make the flock thing?
intuitive vs goal-led	***	[IA1] "your concept for a film needed to have theoretical motivation, conscious aesthetics and a lot of planning … I came from a position of being able to strongly conceptualise things" + "what I'm trying to do in my own artistic development … is learn more intuitive, iterative, work alone playful type practise" + "I can't stand around and wait for people to make ... mistakes that 20 years ago I knew how to avoid by planning"	see also media production. [IA1] has little respect for naïve exploration
expanding beyond original concept	*	[IA2] "I love that sometimes when a project emerges to a certain stage and it becomes evident that something anticipated is actually the strongest part of the project."	
Relating to Technology			
interaction design	*	[IA1] "I should try and have some sense of the information architecture as well, so that could be discussed in the light of the interactivity design as well"	Higher and lower levels of interaction design - "gui will take care of itself" - tony and her mob have a much broader idea. Maybe [IA1] latches onto the broad idea and leaves the gui to the programmer. See communicating metaphorically
being aware of potential	***	[IA1] "sometimes it's difficult for someone like me to know what environment it will finally be done in … you're going to go round in circles until someone says OK we've got to try in this environment now … you often don't have the luxury to play around" [IA2] "it's a process of testing hunches with a bunch of people some of whom have technical abilities" "my naivety actually sometimes might be beneficial because one gets into habits of saying I know what's possible, I know what's not possible" + "I find I have a better understanding of what the algorithm can make happen"	compare [IT3]/[IA2]/[IA1] naïve disinterest in programming - naivety complements Technologist's expertise vs being better artist if you know more. Relate to naïve art "refreshing vision". Toys help being aware of potential. See also "understanding the language of the algorithm"
considering Technology as means to an end	*	[IA1] "Support and help develop and reflect ideas"	
show control	*	[IA2]	This is related to [IT4] - more about control than grand beauty
algorithm like a recipe	*	[IA2]	it isn't really - naïve understanding of structures. This belongs elsewhere.

impact on the arts	**	[IA1] "specificities about computational processes… which have deeply imapcted on, particularly time-based art but also 2D arts as well" "comptational process have made us rethink linearity" [IA2] "Once I understood how the computer could be used for this systems dynamics sort of work I fell out of love with filmmaking"	2 reasons - see below. Artist has latched onto linearity as (sole?) power of computation? Like [IT4] saying interactive art is more about control than abstraction
eases labour-intensive art	*	[IA1] eases labour-intensive art	servitude
Relating to Programming			
aware of difficulties	*	[IA1] "I've had situations in the past where programmers have said to me I don't know whether to go with an XML/SVG sequence, and I'm like in theory I can ask questions but I can't help you I don't have the expertise"	
specificity/concreteness	**	[IA1] "I'm not someone who looks behind and goes oh what does this line mean" [IA2] "those settings are for everyone. Really that's a way to gather around something concrete"	Concrete - toy is a shared object, that doesn' encapsulate every nuance of meaning, but can be interpreted by indiviuals in terms of their own structures of meanings.
[Agile Programming] straddling divide between holistic understanding of eventual delivery and addressing immediate needs	*	[IA2] on agile programming.	[IT4]'s gulf of interpretation. See [IA1] "hardest thing about working with programmers"
disinterest, vs engagement in concepts vs enagement in everything	**	[IA1] "I really like to understand the concepts underlying it … I really enjoyed the 'brain question' … what are the relationships between these two systems and what does it mean in terms of audience effect" [IA2] "I'm intellectually and intuitively really interested in it because I'm interested in most methods of heuristics, of finding out, discovering and creating ... therefore I'm very interested in the kind of problem solving, the imagining the programmer does, the kind of creation of possibility. So I love to try to understand what's kind of going on but I have no time or real aptitude for getting in and doing it"	
front-end interest vs back-end interest	*	[IA1] "is this going to impact on the front end, the functionality of the interaction, or is this a back end thing? And like no it's a back end thing … sometimes I get frustrated with my own lack of knowledge because I can't help with that sort of thing"	So it's not lack of interest in back end, it could be lack of ability to help.
desire to program	*	[IA1] "[I didn't play with Max patches] terribly, and I would, especially the image studio patches ... I could get a lot more instant response"	Toys: instant response

Appendix 8:
Transcript of Interview with George Khut

This appendix is on the accompanying CD

Appendix 9: Excerpt From George Khut's Exegesis

This appendix is on the accompanying CD.

Appendix 10: Artist Biographies

This appendix is on the accompanying CD.

References

Adobe. (2004). Director MX 2004: Adobe.

Adobe. (2005). Flash Professional 8: Adobe.

Amabile, T. M. (1983). *The social psychology of creativity*. New York: Springer-Verlag.

Amabile, T. M. (1987). The motivation to be creative. In S. Isaksen (Ed.), *Frontiers in creativity research: beyond the basics*. Buffalo, NY: Bearly.

Amabile, T. M. (1996). *Creativity in Context: Update to the social psychology of creativity*: Westview Press.

Amabile, T. M., Patterson, C., Mueller, J., Wojcik, T., Odomirok, P., Marsh, M., et al. (2001). Academic-Practitioner Collaboration in Management Research: A Case of Cross-Profession Collaboration. *Academy of Management Journal, 44*(2), 418-431.

Andersen, A. (2002). On Software Design and the Missing Revolution. 2003

Anonymous. (c. 2005). Never-ending fall.

Babchuk, W. A. (1996). *Glaser or Strauss? Grounded Theory and Adult Education*. Paper presented at the Midwest Research-To-Practice Conference in Adult, Continuing and Community Education, Michigan State University, East Lansing, Michigan.

Baecker, R., DiGiano, C., & Marcus, A. (1997). Software Visualization for Debugging. *Communications of the ACM, 40*(4).

Baker-Sennett, J., & Ceci, S. (1996). Clue-efficiency and insight: Unveiling the mystery of inductive leaps. *Journal of Creative Behaviour, 30*, 153-172.

Balaban, M., Barzilay, E., & Elhadad, M. (2002). Abstraction as a Means for End-User Computing in Creative Applications. *IEEE Transactions on Systems, Man, and Cybernetics, 32*(6), 640-653.

Baldwin, N. (1988). *Man Ray: American Artist*. New York: Clarkson Potter.

Barron, F., & Harrington, D. M. (1981). Creativity, Intelligence and Personality. *Annual Review of Psychology, 32*, 439-476.

Barron, R. (1988). Putting Creativity to Work. In R. J. Sternberg (Ed.), *The nature of creativity* (pp. 76-98): Cambridge University Press.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*: Addison-Wesley Publishing Company.

Boden, M. A. (2004). *The Creative Mind: Myths and Mechanisms* (Second Edition ed.). London: Routledge.

Boden, M. A. (Ed.). (1994). *Dimensions of Creativity*. Cambridge, MA, USA: Massachusetts Institute of Technology.

Candy, L., & Edmonds, E. A. (2002). *Explorations in Art and Technology*: Springer.

Candy, L., & Hori, K. (2003). The digital muse: HCI in support of creativity: "creativity and cognition" comes of age: towards a new discipline. *ACM Interactions, 10*(4), 44-54.

Candy, L., & Kelly, A. (2000a). *COSTART Project: Case Report No. 1*. Loughborough, UK: Loughborough University.

Candy, L., & Kelly, A. (2000b). *COSTART Project: Case Report No. 2*. Loughborough, UK: Loughborough University.

Candy, L., & Kelly, A. (2000c). *COSTART Project: Case Report No. 3*. Loughborough, UK: Loughborough University.

Candy, L., & Kelly, A. (2000d). *COSTART Project: Case Report No. 4*. Loughborough, UK: Loughborough University.

Candy, L., & Kelly, A. (2000e). *COSTART Project: Case Report No. 5*. Loughborough, UK: Loughborough University.

Candy, L., & Kelly, A. (2000f). *COSTART Project: Case Report No. 6*. Loughborough, UK: Loughborough University.

Candy, L., & Kelly, A. (2000g). *COSTART Project: Case Report No. 7*. Loughborough, UK: Loughborough University.

Chun, W. H. K. (2005). On Software, or the Persistence of Visual Knowledge. *Grey Room, 18*(Winter 2005).

Cohen, H. (1999). Colouring Without Seeing: a Problem in Machine Creativity. *available from the author*.

Collins, M. A., & Amabile, T. M. (1999). Motivation and Creativity. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 297-312). Cambridge: Cambridge University Press.

Cornock, S., & Edmonds, E. (1973). The Creative Process Where The Artist Is Amplified Or Superseded By The Computer. *Leonardo*(6), 11-16.

Costagliola, G., Delucia, A., Orefece, S., & Polese, G. (2002). A Classification Framework to Support the Design of Visual Languages. *Journal of Visual Languages and Computation, 13*, 573-600.

Couger, D. (1996). *Creativity and Innovation in Information Systems Organizations*. Danvers, MA: Boyd and Fraser.

Craft, B., & Cairns, P. (2005). *Beyond Guidelines: What Can We Learn from the Visual Information Seeking Mantra?* Paper presented at the IV05, London, UK.

Creative Industries Economic Estimates Statistical Bulletin (2005), Department of Culture, Media and Sports, UK [Electronic

(2005).Version] accessed at http://www.culture.gov.uk/NR/rdonlyres/8B1842A1-71D0-464C-9CCA-CD1C52A4D4E1/0/CIEconomicEstimatesREVISED24OCT.pdf.

Crotty, M. (1998). *The foundations of social research : meaning and perspective in the research process*. St Leonards, N.S.W.: Allen & Unwin.

Csikszentmihalyi, M. (1999a). *Creativity: Flow and the Psychology of Discovery and Invention*. New York: HarperCollins Publishers.

Csikszentmihalyi, M. (1999b). Implications of a Systems Perspective for the Study of Creativity. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 313-337). Cambridge: Cambridge University Press.

Davies, C. (Artist). (1995). *Osmose* [virtual reality headset and motion-sensing vest].

Desain, P., & Honing, H. (1993). The Mins of Max. *Computer Music Journal, 17*(2), 3-11.

Dijkstra, E. W. (c1997). The Next Fifty Years.

Domingue, J., & Mulholland, P. (1997). Fostering debugging communities on the Web. *Communications of the ACM, 40*(4).

Dourish, P. (2001). *Where The Action Is: The Foundations of Embodied Interaction*: MIT Press.

Dreyfus, H. L. (2001). *On The Internet: Thinking in Action*. New York: Routledge.

Droste, M., Hahn, P., Hintz, K., Mees, B., Weber, K., Wolsdorff, C., et al. (c. 2000). The "Light-Space modulator" by László Moholy-Nagy. http://www.bauhaus.de/english/bauhaus1919/kunst/kunst_modulator.htm. Retrieved 28 October, 2005

Edmonds, E., Candy, L., Fell, M., Knott, R., Pauletto, S., & Weakley, A. (2003). *Developing Interactive Art Using Visual Programming.* Paper presented at the HCI International 2003, Crete, Greece.

Edmonds, E., Candy, L., Fell, M., Pauletto, S., & Weakley, A. (2005). The Studio as Laboratory: Combining Creative Practice and Digital Technology Research. *International Journal of Human Computer Studies, 63*(4), 452-481.

Edmonds, E., & Muller, L. (2006). On Creative Engagement. *Visual Communications, 5*.

Edmonds, E. A., Everitt, D., Macaulay, M., & Turner, G. (2004). On Physiological Computing with an Application in Interactive Art. *Interacting with Computers, 16*(5), 897-915.

Ehn, P. (1988). *Work-Oriented Design of Computer Artifacts*. Stockholm: Arbetslivscentrum.

Eisenstadt, M. (1997). "My hairiest bug" war stories. *Communications of the ACM, 40*(4).

Elrad, T., Filman, R. E., & Bader, A. (2001). Aspect-oriented programming: Introduction. *Communications of the ACM, 44* (10), 28-32.

Everitt, D. (2002). The Artist as Digital Explorer. In L. Candy & E. A. Edmonds (Eds.), *Explorations in Art and Technology*. London: Springer.

Everitt, D., & Turner, G. (Artist). (1999). *cubeLife* [Internet-Resident Artwork].

Everitt, D., Turner, G., Quantrill, M., & Robson, J. (2002). *Arts and Disability Interfaces: new technology, disabled artists and audiences*: The Arts Council of England.

Feist, G. J. (1999). The Influence of Personality on Artistic and Scientific Creativity. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 273-296). Cambridge: Cambridge University Press.

Fels, S. (Artist). (1999). *Iamascope* [Multimedia Installation].

Fischer, G. (2001). *Communities of Interest: Learning through the Interaction of Multiple Knowledge Systems.* Paper presented at the IRIS'24, Norway.

Fischer, G. (2002). Beyond 'Couch Potatoes': From Consumers to Designers and Active Contributors. *FirstMonday, 7*(12).

Fischer, G. (2005a). Challenges for Future Research Activities and Projects focused on "Software Tools and Socio-Techical Environments to Enhance Creativity". In B. Shneiderman, G. Fischer, M. Czerwinski, B. Myers & M. Resnick (Eds.), *Creativity Support Tools: A workshop sponsored by the National Science Foundation*.

Fischer, G. (2005b). Creativity and Distributed Intelligence. In B. Shneiderman, G. Fischer, M. Czerwinski, B. Myers & M. Resnick (Eds.), *Creativity Support Tools: A workshop sponsored by the National Science Foundation*.

Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A. G., & Mehandjiev, N. (2004). Meta-Design: A Manifesto for End-User Development. *Communications of the ACM, 47*(9), 33-37.

Fischer, G., & Ostwald, J. (2003). Knowledge Communication in Design Communities. In R. Bromme, F. Hesse & H. Spada (Eds.), *Barriers and Biases in Computer-Mediated Knowledge Communication* (pp. 1-32): Kluwer Academic Publishers.

Flanagan, M., & Perlin, K. (2004). *Endpapers: collaboration, process and code.* Paper presented at the ISEA2004, Helsinki, Tallin.

Florida, R. (2002). *The RIse of the Creative Class and How It's Transforming Work, Leisure, Community and Everyday Life*. New York, US: Basic Books.

Foddy, W. H. (1992). *Constructing questions for interviews and questionnaires : theory and practice in social research*. Cambridge, England ; Melbourne: Cambridge University Press.

Fraser, D. (1999). *QSR NUD*IST Vivo: Reference Guide*. Melbourne: Qualitative Solutions and Research Pty.

Frenkel, K. (1994). A Conversation with Alan. *Interactions*(April 1994).

Fry, C. (1997). Programming on an Already Full Brain. *Communications of the ACM, 40*(4).

Gardner, H. (1993). Seven creators of the modern era. In J. Brockman (Ed.), *Creativity* (pp. 28-47). New York: Simon & Schuster.

Gaver, W. W., Beaver, J., & Benford, S. (2003). *Ambiguity as a Resource for Design*. Paper presented at the ACM Conference on Human Factors in Computing Systems, Ft. Lauderdale, Florida, USA.

Giaccardi, E., & Fischer, G. (2005). *Creativity and Evolution: A metadesign perspective.* Paper presented at the European Academy of Design (EAD-6) Conference, Bremen, Germany.

Glaser, B. G. (1992). *Basics of Grounded Theory Analysis*: Sociology Press.

Glaser, B. G. (1998). *Doing grounded theory : issues and discussions*. Mill Valley, Calif.: Sociology Press.

Glaser, B. G., & Strauss, A. L. (1967). *The Discovery of Grounded Theory: strategies for qualitative research*. Hawthorne, N.Y: Aldine Publishing Company.

Glaser, R., & Chi, M. T. H. (1988). *The Nature of Expertise*.

Good, J. (1999). *VPLs and Novice Program Comprehension: How do Different Languages Compare?* Paper presented at the IEEE Symposium on Visual Languages VL '99.

Graham, P. (2002). Revenge of the Nerds [Electronic Version]. http://www.paulgraham.com/icad.html. Retrieved 24 November 2005.

Graham, P. (2003). Fortran I. http://www.paulgraham.com/history.html. Retrieved 24 February, 2004

Gruber, H. E. (1989). The Evolving Systems approach to creative work. In D. B. Wallace & H. E. Gruber (Eds.), *Creative People at Work: Twelve Cognitive Case Studies* (pp. 3-24). New York: Oxford University Press.

Guildford, J. P. (1950). Creativity. *American Psychologist, 5*, 444-454.

Guindon, R., & Curtis, B. (1988). Control of cognitive processes during software design: What tools are needed. *CHI'88: Conference Proceedings: Special Issue of the ACM/SIGCHI Bulletin*, 263-268.

Guindon, R., Krasner, H., & Curtis, B. (1987). *Cognitive Processes in Software Design*. Paper presented at the 2nd IFIP Conference on Human Computer Interaction--INTERACT'87, North Holland.

Heemskerk, J., & Paesmans, D. (Artist). (2005). *jodi.org* [Website].

Heidegger, M. (1962). *Being and Time*: Blackwell.

Hewett, T., Czerwinski, M., Terry, M., Nunamaker, J., Candy, L., Kules, B., et al. (2005). Creativity Support Tool Evaluation Methods and Metrics. In B. Shneiderman, G. Fischer, M. Czerwinski, B. Myers & M. Resnick (Eds.), *Creativity Support Tools: A workshop sponsored by the National Science Foundation*.

Hewett, T. (2002). An Observer's Reflections: The Artist Considered as Expert. In L. Candy & E. A. Edmonds (Eds.), *Explorations in Art and Technology*. London: Springer.

Holstein, J. A., & Miller, G. (1993). *Reconsidering social constructionism : debates in social problems theory*. New York: Aldine de Gruyter.

Höök, K., Sengers, P., & Andersson, G. (2003). Sense and Sensibility: Evaluation and Interactive Art. *CHI Letters, 5*(1), 241-248.

Ingalls, D., Kaehler, T., Maloney, J., Wallace, S., & Kay, A. (1997). *Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself.* Paper presented at the OOPSLA'97 Conference, Atlanta, Georgia.

Ingalls, D. H. H. (1981). Design Principles Behind Smalltalk. BYTE Magazine,, August 1981.

Isaksen, S. G., & Treffinger, D. J. (1985). *Creative Problem Solving: The Basic Course*. Buffalo, NY, USA: Bearly Ltd.

Isaksen, S. G., & Treffinger, D. J. (2004). Celebrating 50 years of reflective practice: Versions of creative problem solving. *Journal of Creative Behaviour, 38*(2), 75-101.

Jarry, A. (1980). Exploits and Opinions of Dr Faustroll, 'Pataphysician: a Neo-Scientific Novel. In R. Shattuck & S. W. Taylor (Eds.), *Selected Works of Alfred Jarry*. London: Eyre Methuen Ltd.

Jausovec, N., & Bakracevic, K. (1995). What can heart rate tell us about the creative process? *Creativity Research Journal, 8*, 11-24.

Jennings, P., & Giaccardi, E. (2005). Creativity Support Tools for *and* by the New Media Arts Community. In B. Shneiderman, G. Fischer, M. Czerwinski, B. A. Myers & M. Resnick (Eds.), *Creativity Support Tools* (pp. 37-52).

John-Steiner, V. (2000). *Creative Collaboration*: Oxford University Press.

Kan, M. M. (2002). Grounded Theory Methodology: A process map (private communication).

Kahn, K. (1996). ToonTalk™—An Animated Programming Environment for Children. *Journal of Visual Languages and Computation,* 7(2).

Kautz, K. (2004). *The Enactment of Methodology - The Case of Developing a Multimedia Information System.* Paper presented at the International Conference of Information Systems, Washington DC, USA.

Kautz, K., Hansen, B., & Jacobsen, D. (2004). The Utilization of Information Systems Development Methodologies in Practice. *Journal of Information Technology Cases and Applications, 6*(4).

Kay, A. (1993). The Early History of Smalltalk. *ACM SIGPLAN Notices, 28*(3), 69-95.

Kay, A., & Goldberg, A. (1977). Personal Dynamic Media. *Computer, 10*(3), 31-41.

Khut, G. P., & Muller, L. (2005). *Evolving Creative Practice: A Reflection on Working with Audience Experience in Cardiomorphologies.* Paper presented at the Vital Signs: Creative Practice & New Media Now, Melbourne, Australia.

Khut, G. (2006). *Development and Evaluation of Participant-Centred Biofeedback Artworks*. University of Western Sydney, Sydney, Australia.

Knuth, D. E. (1984). Literate Programming. *The Computer Journal, 27* (2), 97-111.

Larkin, M. (2004). Conceptual issues in psychology (lecture notes). Department of Psychology, De Montfort University.

Levin, G., Ward, A., lia, & meta. (2001). *4x4 Generative Design: Beyond Photoshop*: Friends Of Ed.

Levitt, T. (2002/1963). Creativity is Not Enough. *Harvard Business Review special issue on The Innovative Enterprise*(August 2002), 137-145.

Lieberman, H. (1995). *The Visual Language of Experts in Graphic Design*. Paper presented at the IEEE Symposium on Visual Languages, Darmstadt, Germany.

Lieberman, H. (1997). The Debugging Scandal and What to Do About It. *Communications of the ACM, 40*(4).

Lieberman, H. (2001a). Interfaces that Give and Take Advice. In J. Carroll (Ed.), *Human-Computer Interaction for the New Millenium* (pp. 475-485): ACM Press/Addison-Wesley.

Lieberman, H. (2001b). *Your Wish is My Command: Programming By Example*: Morgan Kaufmann.

Lieberman, H., & Fry, C. (1997). ZStep 95, A Reversible, Animated Source Code Stepper. In J. Stasko, J. Domingue, M. Brown & B. Price (Eds.), *Software Visualization: Programming as a Multimedia Experience*. Cambridge, MA,: MIT Press.

Liu, H., & Lieberman, H. (2005). *Programmatic Semantics for Natural Language Interfaces.* Paper presented at the ACM Conference on Human Factors in Computing Systems, CHI 2005, Portland, OR, USA.

Loseke, D. R. (1999). *Thinking about social problems : an introduction to constructionist perspectives*. New York: Aldine de Gruyter.

Lubart, T. I. (1990). Creativity and Cross-Cultural Variation. *International Journal of Psychology, 25*(1), 39-59.

Lubart, T. I. (1999). Creativity Across Cultures. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 339-350). Cambridge: Cambridge University Press.

Lubart, T. I., & Sternberg, R. J. (1995). An investment approach to creativity: theory and data. In S. M. Smith, T. B. Ward & R. A. Finke (Eds.), *The creative cognition approach* (pp. 269-302). Cambridge, MA: MIT Press.

MacLean, A., Carter, K., Lövstrand, L., & Moran, T. (1990). *User-Tailorable Systems: Pressing the Issues with Buttons.* Paper presented at the CHI '90 Proceedings.

Maduro, R. (1976). *Artistic Creativity in a Brahmin painter community.* Unpublished Research Monograph 14, Center for South and Southeast Asia Studies, University of California, Berkeley.

Maeda, J. (1999). *Design By Numbers*: MIT Press.

Maeda, J., & Burns, R. (2004). *Creative code*. London: Thames & Hudson.

Mamykina, L., Candy, L., & Edmonds, E. (2002). Collaborative Creativity. *Communications of the ACM, 45*(10), 96-99.

Masayasu, I. (2005). HTML/XHTML: W3C.

Mateas, M. (2005). Procedural Literacy: Educating the New Media Practitioner. *On The Horizon. Special Issue. Future of Games, Simulations and Interactive Media in Learning Contexts, 13*(1).

Mathes, A. (2004). Folksonomies—Cooperative Classification and Communication Through Shared Metadata. Graduate School of Library and Information Science, University of Illinois Urbana-Champaign.

Mayer, R. E. (1999). Fifty Years of Creativity Research. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 449-460). Cambridge: Cambridge University Press.

McCarthy, J. (1981). History of LISP. In R. Wexelblat (Ed.), *History of Programming Languages*: Academic Press.

McCormack, J. (Artist). (1994). *Wild. An Interactive Computer Installation*

McKenzie, M. (1994). *The Amazing Grace Hopper.* Paper presented at the Grace Hopper Celebration of Women in Computing Conference 1994.

McLuhan, M. (1964). *Understanding media : the extensions of man*. New York: McGraw-Hill.

Moher, T. G. (1988). PROVIDE: A Process Visualization and Debugging Environment. *IEEE Transactions on Software Engineering, SE-14*(6), 849-857.

Muller, L. (2004). The public face of online interaction. *Realtime Arts*.

Muller, L. (2005). *Transforming Mirrors: Understanding the audience experience of interactive art*. Sydney, Australia: University of Technology, Sydney.

Muller, L., & Edmonds, E. (2006). *Living Laboratories: Making and Curating Interactive Art*. Paper presented at the SIGGRAPH, Boston.

Muller, L., Turner, G., Khut, G., & Edmonds, E. (2006). *Creating Affective Visualisations for a Physiologically Interactive Artwork*. Paper presented at the IV06, London, UK.

Murray, B., Candy, L., & Edmonds, E. (1996). User Centred Complex System Design: Combining Strategy, Methods and Front End Technology. In *Critical Issues in User Interface Systems Engineering* (pp. 169-188). London, UK: Springer-Verlag.

Myers, B. A. (1990). Taxonomies of Visual Programming and Program Visualization. *Visual Languages and Computing, 1*(1), 97-123.

Myers, B. (c1999). Usability Issues in Programming Languages. from http://www.cs.cmu.edu/~NatProg/langeval.html

Nardi, B. (1993). *A Small Matter of Programming*. Cambridge, MA: MIT Press.

NCH Swift Sound. (2004). Express Scribe.

Negroponte, N. P. (1995). *Being Digital*. New York: Alfred A. Knopf.

Neumark, N. (2001). *E/motional Machines: Ésprit de Corps.* Paper presented at the Affective Encounters: rethinking embodiment in feminist media studies, Turku.

Neumark, N., & Miranda, M. (2003). Institute for the Study of Perpetual Emotion. http://turbulence.org/studios/rumor/emotion/. Retrieved 18 July, 2004

Nickerson, R. S. (1999). Enhancing Creativity. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 392-431). Cambridge: Cambridge University Press.

Nightingale, D., & Crobmie, J. (Eds.). (1999). *Social Constructionist Psychology: a critical analysis of theory and practice*. Buckingham, UK: Open University Press.

Norman, D. A., & Draper, S. W. (Eds.). (1986). *User-Centered System Design, New Perspectives on Human–Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Assoc Inc.

O'Brien, R. (1998). Um exame da abordagem metodológica da pesquisa ação [An Overview of the Methodological Approach of Action Research]. In R. Richardson (Ed.), *Teoria e Prática da Pesquisa Ação [Theory and Practice of Action Research].* João Pessoa, Brazil.

Oliver, J., Chatterton, C., Blundell, A., & Cannon, R. (Artist). (1998). *acmipark* [Virtual Environment].

Osborn, A. F. (1953). *Applied Imagination*. New York: Scribner's.

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research, 60*(1), 65-89.

Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas*. New York: Basic Books.

Papert, S., & Harel, I. (Eds.). (1993). *Constructionism*. Norwood, NJ: Ablex Publishing Corporation.

Parnes, S. J. (1967). *Creative Behaviour Guidebook*. New York, NY, USA: Charles Scribner's Sons.

Paul, C. (2002). *CODeDOC* exhibition. http://www.whitney.org/artport/commissions/codedoc/

Picard, R. (1997). Does HAL Cry Digital Tears? Emotions and Computers. In D. Stork (Ed.), *HAL's Legacy: 2001's Computer as Dream and Reality* (pp. 279-304). Cambridge: MIT Press.

Pinckard, J. (2003). game girl advance: The Destiny of Games. http://www.gamegirladvance.com/archives/2003/03/04/the_destiny_of_games.html. Retrieved 8 December, 2004

Porteous, M., Kirakowski, J., & Corbett, M. (1993). *SUMI User Handbook*. Cork, Ireland: Human Factors Research Group, University College Cork.

Potter, J. (1996). *Representing reality : discourse, rhetoric and social construction*. London ; Thousand Oaks, CA: Sage.

Proctor, T., Tan, K. H., & Fuse, K. (2004). Cracking the Incremental Paradigm of Japanese Creativity. *Creativity and Innovation Management, 13*(4), 207-215.

Puckette, M. (1996). *Pure data: Another integrated computer music environment.* Paper presented at the Second Intercollege Computer Music Concerts, Tachikawa, Japan.

Puckette, M. (2002). Max at Seventeen. *Computer Music Journal, 26*(4), 31-43.

Ramachandran, V. S., & Hubbard, E. (2003). Hearing Colors, Tasting Shapes. *Scientific American, 288*(5), 42-49.

Ramachandran, V. S., & Hubbard, E. M. (2001a). Psychophysical investigations in to the neural basis of synaesthesia. *Proceedings of the Royal Society, 268*, 979-983.

Ramachandran, V. S., & Hubbard, E. M. (2001b). Synaesthesia--a window into perception, thought and language. *Journal of Consciousness Studies, 8*, 3-34.

Reas, C. (2004). *Programming Media.* Paper presented at the PixelRaiders 2 Conference, Sheffield, UK.

Reas, C., & Puckette, M. (2004). *Panel: Who owns our (software) culture?* Paper presented at ISEA2004, Helsinki, Tallinn.

Repenning, A., & Perrone, C. (2000). Programming by Analogous Examples. *Communications of the ACM, 43*(3), 90-97.

Resnick, L. B., Levine, J. M., & Teasley, S. D. (1991). *Perspectives on Socially Shared Cognition*. Washington, DC: American Psychological Association.

Resnick, M., Myers, B. A., Nakakoji, K., Shneiderman, B., Pausch, R., Selker, T., et al. (2005). Design Principles for Tools to Support Creative Thinking. In B. Shneiderman, G. Fischer, M. Czerwinski, B. Myers & M. Resnick (Eds.), *Creativity Support Tools: A workshop sponsored by the National Science Foundation*.

Rettig, M. (2003). Interaction Design History in a teeny little nutshell http://www.marcrettig.com/writings/rettig.interactionDesignHistory.2.03.pdf.

Ricadela, A. (2002). Retooling The Programmers. *Information Week*, Nov 18, 2002.

Richards, K. (2005). Let the Body Navigate: An interview with George Khut. *RealTime, 66*.

Rittenbruch, M. M., Gregor, Ward, N.l, Mansfield, T., Bartenstein, D. (2002). *Extreme Participation - Moving Extreme Programming Towards Participatory Design.* Paper presented at the Seventh Biennial Participatory Design Conference, Malmö, Sweden.

Rokeby, D. (1996). Transforming Mirrors: Subjectivity and Control in Interactive Media. In S. Penny (Ed.), *Critical Issues in Interactive Media*: SUNY press.

Rokeby, D. (1998). The Construction of Experience: Interface as Content. In C. Dodsworth (Ed.), *Digital Illusion: Entertaining the Future with High Technology*: Addison-Wesley.

Runco, M. A., & Sakamoto, S. O. (1999). Experimental Studies of Creativity. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 62-92). Cambridge: Cambridge University Press.

Sanford, L. (1997). *ENIAC Programmers*, Women In Technology International http://www.witi.com/center/witimuseum/halloffame/1997/eniac.shtml. 2003

Sankaran, N. (1995). Looking Back At ENIAC: Computers Hit Half-Century Mark. *The Scientist, 9,* 3.

Schön, D. A. (1983). The Reflective Practitioner: How Professionals Think In Action. New York: Basic Books.

Shanfelder, B. (2005). Audio Recorder (Version 2.2).

Shaw, J., & Pledger, D. (Artist). (2004). *Eavesdrop* [Interactive Cinema Installation].

Shneiderman, B. (1980). *Software Psychology: Human Factors in Computer and Information Systems*. Cambridge, Mass.: Winthrop.

Shneiderman, B. (2002a). Creativity Support Tools: Establishing a framework of activities for creative work. *Communications of the ACM, 45*(10), 116-120.

Shneiderman, B. (2002b). *Leonardo's Laptop*.

Shneiderman, B., Fischer, G., Czerwinski, M., Myers, B., & Resnick, M. (2005). *Creativity Support Tools: A workshop sponsored by the National Science Foundation*.

Shneiderman, B., Mayer, R., McKay, D., & Heller, P. (1977). Experimental Investigations of the Utility of Detailed

Flowcharts in Programming. *Communications of the ACM, 20* (6), 373-381.

Silver, H. R. (1981). Calculating Risks: The socioeconomic foundations of aesthetic innovation in an Ashanti carving community. *Ethnology, 20*(2), 101-114.

Smith, K. (2004). *From transmission to multiplicity: interactive art installations as a site for research*. Paper presented at the ACM international conference on Multimedia, New York, NY, USA.

Smith, R. B., Wolczko, M., & Ungar, D. (1997). From Kansas to Oz: collaborative debugging when a shared world breaks. *Communications of the ACM, 40*(4).

Stephenson, N. (1999). In the Beginning was the Command Line. http://www.cryptonomicon.com/beginning.html.

Sternberg, R. J. (1999). *Handbook of Creativity*. Cambridge, UK: Cambridge University Press.

Sternberg, R. J., & Lubart, T. I. (1995). *Defying the Crowd: Cultivating Creativity in a Culture of Conformity*. New York: Free Press.

Strauss, A. L., & Corbin, J. (1998). *Basics of qualitative research: Techniques and procedures for developing grounded theory* (2nd ed.). Thousand Oaks, CA: SAGE Publications Ltd.

Suchman, L. (1987). *Plans and Situated Actions: The problem of human/machine communication*: Cambridge University Press.

Suchman, L. (2002). Replicants and Irreductions: Affective encounters at the interface. *European Association for the Study of Science and Technology (EASST)*.

Suchman, L. (2003). Embodied Agencies at the Interface.

Sun Microsystems. (2005). Java 2 Platform, Standard Edition (J2SE): Sun Microsystems.

Suthers, D. (2003). Technology Affordances for Intersubjective Learning, and How They May Be Exploited. In R. Bromme,

F. Hesse & H. Spada (Eds.), *Barriers and Biases in Computer-Mediated Knowledge Communication* (pp. 1-32): Kluwer Academic Publishers.

Terry, M. (2005). *Set-Based User Interaction*. Unpublished Ph.D. Thesis, Georgia Institute of Technology, Georgia.

Terry, M., & Mynatt, E. D. (2002). *Recognizing Creative Needs in User Interface Design*. Paper presented at the Creativity and Cognition Conference 2002, Loughborough, UK.

Thomas, C., & Bevan, N. (Eds.). (1995). *Usability Context Analyisi: A Practical Guide*. Middlesex, UK: National Physical Laboratory.

Torrance, E. P. (1987). Recent trends in teaching children and adults to think creatively. In S. G. Isaksen (Ed.), *Frontiers of creativity research: Beyond the basigs* (pp. 204-215). Buffalo, NY, USA: Bearly Ltd.

Treadwell, A. *Virtual Transcendence: An exploration of the 'mystical' experience professed by those having experienced certain virtual reality environments.*, Oxford Brookes, Oxford, UK.

Turing, A. M. (1937). On Computable Numbers with an Application to the Entscheidungsproblem. *Proc. London Math. Soc., 42*, 230-265.

Turkle, S. (1984). *The Second Self: Computers and the Human Spirit*. London, UK: Granada Publishing Ltd.

Turner, G. (2002). *cubeLife: the evolution of an Internet-resident artwork*. Unpublished Master's Thesis, Loughborough University, Loughborough, UK.

Turner, G., & Edmonds, E. A. (2003). *Towards a Supportive Technological Environment for Digital Art*. Paper presented at the OzCHI 2003: New directions in interaction, information environments, media and technology, Brisbane, Australia.

Turner, G., Neumark, N., Miranda, M., & Weakley, A. (2004). *Uncanny Interaction: A Digital Medium for Networked E.motion*. Paper presented at the Interaction: Systems, Practice and Theory, Sydney, Australia.

Ullman, D. G., Dietterich, T. G., & Stauffer, L. (1988). A Model of the Mechanical Design Process Based on Empirical Data. *AI EDAM, 2*(1), 33-52.

von Hippel, E. (2002). Horizontal innovation networks - by and for users. MIT Sloan School of Management

von Neumann, J. (1945). First Draft Report on the EDVAC. *IEEE Annals of the History of Computing, 15*(4), 27-75.

Ward, A. (Artist). (2002). *AutoIllustrator* [Computer Application].

Ward, T. B., Smith, S. M., & Finke, R. A. (1999). Creative Cognition. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 189-212). Cambridge: Cambridge University Press.

Weakley, A., Johnston, A., Edmonds, E., & Turner, G. (2005, August 11, 2005). *Creative Collaboration: Communication Translation and Generation in the Development of a Computer-Based Artwork.* Paper presented at the HCI International, Las Vegas.

Wehner, L., Csikszentmihalyi, M., & Magyari-Beck, I. (1991). Current Approaches used in Studying Creativity: An exploratory investigation. *Creativity Research Journal, 4*(3), 261-271.

Weisberg, R. W. (1999). Creativity and Knowledge: A Challenge to Theories. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 226-250). Cambridge: Cambridge University Press.

Weisberg, R. W., & Alba, J. W. (1981). An examination of the alleged role of fixation in the solution of several insight problems. *Journal of Experimental Psychology: General, 110*, 169-192.

Williams, W. M., & Yang, L. T. (1999). Organizational Creativity. In R. J. Sternberg (Ed.), *Handbook of Creativity* (pp. 373-391). Cambridge: Cambridge University Press.

Wilson, R. A., & Keil, F. C. (Eds.). (2001). *The MIT Encyclopedia of the Cognitive Sciences*: MIT Press.

Wilson, S. (2002). *Information Arts: Intersections of Art, Science and Technology*. Cambridge, Massachusetts: The MIT Press.

Winograd, T. (1995). Heidegger and the Design of Computer Systems. In A. Feenberg & A. Hannay (Eds.), *Technology and the Politics of Knowledge*: Indiana University Press.