Gradual Structuring: Evolving the Spreadsheet Paradigm for Expressiveness and Learnability

Gary Miller
School of Computing
and Communications
University of Technology Sydney
Email: gary.miller@student.uts.edu.au

Felienne Hermans
Software Engineering Research Group
Delft University of Technology
Email: f.f.j.hermans@tudelft.nl

Robin Braun
School of Computing
and Communications
University of Technology Sydney
Email: Robin.Braun@uts.edu.au

***Abstract*—Spreadsheets are arguably the most used form of programming and are frequently used in higher education to teach fundamental concepts about computation. Their success has shown that they are simple enough for a huge number of end users to learn and use. This is in contrast to traditional programming languages and the high dropout rate from introductory programming and computer science. However in comparison to traditional programming languages and structured modelling, spreadsheets are not expressive, placing a limit on the levels of computational thinking that can be taught using the spreadsheet paradigm. This limitation is imposed by the lack of programming language features and abstractions in the paradigm. Furthermore, more advanced spreadsheet features (e.g. array formulae, lookup formulae, R1C1 syntax) can be difficult to learn and use.**

This paper discusses the idea of adding language features to spreadsheets, enabling the gradual structuring of free-form spreadsheets to more structured models. We propose that this concept is termed Gradual Structuring, and is analogous to the programming language concept of gradual typing. In this analogy, spreadsheets take the place of dynamic programming and structured modelling of static programming. In programming languages, gradual typing allows dynamic programming to be mixed with static programming. It is our contention that dynamic programming is more learnable while static programming is more expressive and abstract. Gradual typing could be used to mitigate the issues in the teaching of traditional programming. Likewise Gradual Structuring can mitigate the conceptual limits that can be taught using current spreadsheets.

The key language feature required to enable Gradual Structuring is the ability to logically group cells together so that a single formula can be applied to the grouped cells. This concept, termed cell grouping diminishes and can even eliminate the need for the ubiquitous and error-prone use of copy-pasted in spreadsheets. Moreover, it makes the structure present in spreadsheet models explicit. Cell grouping requires a cascade of other new languages features. Namely a more expressive referencing style, which in turned requires enabling labels to be moved to the row and column headers, and the hierarchical structuring of these headers. Respectively these language features are termed enhanced referencing and semantic axes.

The ongoing research focusses on the usability and learnability of these language features. Spreadsheet applications exist that contain aspects of the features mentioned. However these applications do not enable Gradual Structuring and have taken a mainly technical, not human behavioural, approach to evolving the spreadsheet.

I. Introduction

The basic features of Spreadsheets as they exist today arguably make a better introductory programming environment for higher education than conventional programming languages [1]. This applies to a broad rage of domains, not just science and engineering. However this does not extend much past introductory programming as the spreadsheet paradigm does not contain the necessary abstractions to comfortably support complex programs.

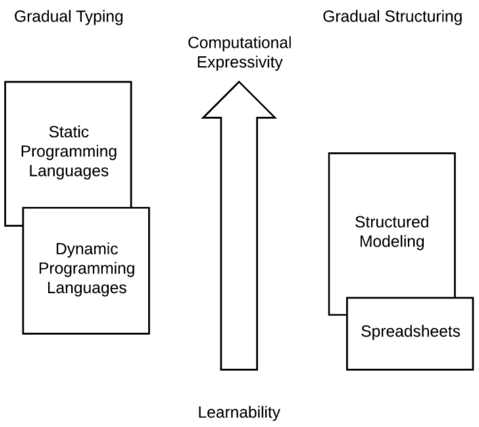

Gradual Typing
Gradual Structuring
Computational Expressivity
Static Programming Languages
Dynamic Programming Languages
Structured Modeling
Spreadsheets
Learnability

Fig. 1: Explanatory Diagram - Relationship Between Gradual Typing, Gradual Structuring, Expressiveness and Learnabilty

It is increasingly necessary for students to take programming subjects. Unfortunately concepts currently learnt from spreadsheets do not transfer to general programming. It is our contention that the spreadsheet paradigm could be evolved to support more complex problems. This would extend the useful domain of the paradigm and hopefully increase the transferability of concepts when learning programming.

Abstractions can be added to the spreadsheet paradigm to increase its expressiveness. It is our contention that these language features can be designed to maintain and improve the learnability of the paradigm. The key concept is to allow people to use the features they find intuitive and provide the scaffolding for them to access the more abstract and expressive features. This is a concept we term Gradual Structuring.

This paper is laid out as follows. Section II draws an analogy between Gradual Structuring and gradual typing, and is a background to issues in spreadsheets and general programming. In section III a motiving example highlighting expressive issues in currently spreadsheet modelling is provided. Section IV proposes requirements for Gradual Structuring, puts forward some language features to enable these and shows how these features can be used to create more structured versions of the motivating example. Section V is a discussion of Gradual Structuring from an expressiveness and learnability perspective. Section VI is on related work, and section VII are concluding remarks.

II. Background

Spreadsheets are outstandingly successful. However there are many obvious issues with spreadsheets. The most visible of which are errors in spreadsheet models. Spreadsheet errors make the general news on a regular basis (e.g. [2]), and research into understanding spreadsheet errors is well advanced [3]. In large part the analogous errors in programming are mitigated by capabilities not present in spreadsheets (e.g. unit tests and compiler errors). These capabilities are enabled by abstractions in these systems (e.g. functions and types).

Spreadsheets do not allow for even the most basic forms of abstractions that exist in most programming systems (e.g. functional decomposition [4]). The trend has been to create trap doors to other environments to gained increased expressiveness. A good example of these trap doors are Microsoft Excel's VBA based macros, Power Query and Power Pivot, which are respectively based on imperative, functional and relational programming paradigms. Learnings in the declarative spreadsheet paradigm does not readily transfer to abilities in these more traditional paradigms. Users are therefore left having to learn multiple programming paradigms rather than continually build on their existing knowledge and skills.

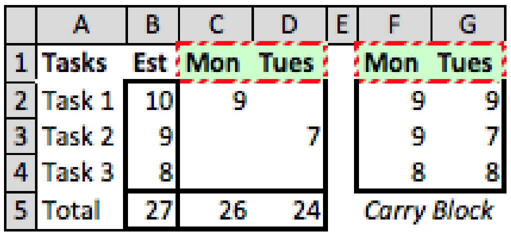
Mastering spreadsheet modelling is also difficult. There are technical gaps between similar pieces of functionality. For example a lookup formula is similar to a plain reference, but is accessed by the use of functions. Mastering cell referencing does not aid in understanding formula based lookup functionality. More importantly model design is difficult. There are only a few weak features that aim in directing and constraining design choices.

One of the most common concepts across programming languages is the idea of variables. A memory location that's contents can be changed. From a cognitive perspective this is a counter intuitive abstraction [5] analogous to the philosophical separation of a name from the object that it represents [6]. Ambler [7] noted that the concept of a variable does not exist in the spreadsheet paradigm.

The use of static verses dynamic typing is much more controversial [8] than variables. Gradual typing seeks to balance the trade-offs presented by the two different approaches [9]. Here languages are extended to support both dynamic and static programming, allowing incremental adding of types to programs. Industry has embraced gradual typing to help solve some of the endemic issues, with large scale systems written in dynamic languages [10], [11].

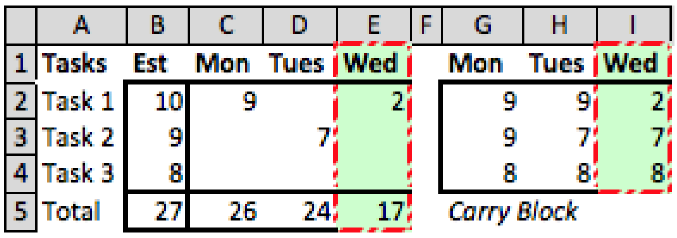
In this paper we draw and analogy between gradual typing and a concept we have termed Gradual Structuring. Here

(a) Repeated Columns



	A	B	C	D	E	F	G
1	Tasks	Est	Mon	Tues		Mon	Tues
2	Task 1	10	9			9	9
3	Task 2	9		7		9	7
4	Task 3	8				8	8
5	Total	27	26	24		Carry Block	

(b) Addition of a day



	A	B	C	D	E	F	G	H	I
1	Tasks	Est	Mon	Tues	Wed		Mon	Tues	Wed
2	Task 1	10	9		2		9	9	2
3	Task 2	9		7			9	7	7
4	Task 3	8					8	8	8
5	Total	27	26	24	17		Carry Block		

(c) Logic Groups - Equivalent Formulae

```
Est Total
=SUM(B2:B4)
Task Totals
=SUM(G2:G4)
Carry Block
=IF(ISBLANK(C2),IF(ISBLANK(F2),$B2,F2),C2)
```

Fig. 2: Repetition Problem - Scrum Burndown - Heavy bordered areas indicate logical groups of cells.

structured modelling, for example relational databased, is analogous to statically typed programming and spreadsheet modelling to dynamically typed systems.

Gradual Structuring proposes a solution that supports the seamless coexistence of free-form and structured modelling. This is achieved by adding language based features to the spreadsheet paradigm. These are carefully selected abstractions that increase expressiveness and maintain usability. In the vein of linguistic abstraction [12], these features are the modelling design patterns of experts.

III. Motivating Example

Copy-paste repetition of formulae is ubiquitous in spreadsheet modelling. Repeated structure in models, e.g. multiple blocks of the same dimensions surrounded by the same labels, are also common in spreadsheets, depending on the problem being modelled. Both of these forms of repetition are sources of error.

The motivating example described in this paper is the use of a Scrum Burndown spreadsheet to address an Agile, processing problem (see Figure 2a). Two forms of redundancy are demonstrated; the copy-paste repetition of formulae, and the repetition of headers in intermediate calculations blocks.

This spreadsheet enables users to estimate and enter the remaining time for a task that they worked on on a particular day as opposed to carrying across estimates for tasks that were not worked on. This requirement mandates an intermediate block to calculate and carry forward the estimates from the last day that the task was worked on. The carry block exhibits repeated headers (see Figure 2a) which are identical to the headers in the input block. Adding an extra day requires the insertion of a new column in both the input block and the carry block and these formulae need to be copy-pasted into the newly created cells (see Figure 2b). Note the repeated column headers.

There are three copy equivalent [13] formulae in this model. The formula in the first cell of each of these is shown in Figure 2c.

As demonstrated in this example, copy-paste appears to have some abstraction properties, as the references differ between the source and destination cells. This is a compensation for the lack of abstraction in the spreadsheet paradigm [14].

IV. Gradual Structuring

A. Requirements

This section consists of the requirements for Gradual Structuring taken from the perspectives of both free-form modelling and structured modelling. Following this the features that enable these requirements are investigated. Using these features the motivating example is refactored into a partially structured model, and then a fully structured model. Finally the consequences of Gradual Structuring are discussed from the perspectives of expressiveness and learnability.

The requirements for Gradual Structuring come from both the spreadsheet paradigm and structured modelling. From the world of spreadsheets comes methodologies, standards, best practise, modelling design patterns and other informal conventions. More formal requirements come from OLAP, multi-dimensional modelling tools (typified by Lotus Improv), relational databases and other programming paradigms.

Spreadsheet Modelling Conventions

The evidence for the desirability for a cell grouping feature comes from a number of sources. Many spreadsheet auditing tools contain a maps report (e.g. Operis Analysis Kit & Spreadsheet Innovations) or distinct formula view (e.g. Spreadsheet Detective) that show the cells that contain the same formula, which in effect should be grouped. Spreadsheet design literature [15] explicitly advocates for the grouping of cells. Academic work on spreadsheet smell detection [16] infers that clusters of cells and uses this in the detection of errors.

The level of structure and grouping used by expert modellers can also be seen in the ModelOff [17] solutions. Observational studies of expert spreadsheet modellers has identified an interesting pattern, which is the extent to which they structure their models. This structuring consists of two parts: that of the layout of their models so that cells having the same formula are contiguous; and the labelling of these blocks to provide information about the blocks. Such structuring enables modellers to operate at the level of groups of cells and using the application of a single formula to the group (e.g. via entering a formula using Ctrl+Enter on selected cells which is equivalent to copy and then paste special formulae). Stronger evidence for this behaviour is the restructuring of formulae to apply to a broader set of cells (e.g. `IF(`*`first column`*`,`*`initial formula`*`,`*`formula`*`)`).

Formal Requirements from Structured Modelling

An appropriate set of requirements for structured modelling can be taken from OLAP [18]. These are:

1) Rich dimensional structuring with hierarchical referencing.
2) Efficient specification of dimensions and calculations.
3) Flexibility.
4) Separation of structure and representation.

OLAP is only one possible source of formal requirements. Traditional relational databases can be considered to be even more structured than OLAP. Multi-dimensional modelling is less structured, but closer to the flexibility of spreadsheets. OLAP is chosen as there is already overlap with structured modelling functionality in spreadsheets (e.g. pivot tables). Although OLAP is a flexible database technology, it does not have the flexibility of spreadsheet modelling as separation of structure and representation is enforced (i.e. formulae cannot be put into arbitrary cells.).

Similar requirements can be found in multi-dimensional modelling applications [19], [20]. Lotus Improv's core requirements were the separation of data, formulae and view.

B. Enabling Features

In this subsection the features that enable Gradual Structuring are defined and discussed.

Cell Grouping

Cell grouping is a feature that allows a single formula to be applied to a group of cells. This abrogates the copy-paste repetition and redundancy described previously. A more expressive enhanced referencing semantic, discussed below, is required to facilitate cell grouping. Enhanced referencing is required for both specifying the cell groups and the references in the formulae. An implementation and concrete example of cell grouping can be found in the Sumwise spreadsheet application [21].

Enhanced Referencing

The key abstraction needed to enable cell grouping and thereby Gradual Structuring is an enhanced referencing semantic. It is designed to be readable as with A1 referencing, yet have the invariant quality of R1C1 referencing, which enables a reference to remain the same when applied to another cell.

Current spreadsheets use A1 referencing whereby the columns are specified by letters and the rows by a number. This is important as there is an alternate and more expressive form of referencing. The R1C1 syntax is available in most spreadsheet applications. Here both columns and rows are specified using numbers. With R1C1 referencing the text representation does not change when copy-pasted.

Before giving some enhanced referencing examples semantic axes are discusses, as enhanced referencing depends on them.

Semantic Axes

Enhanced referencing is based on more expressive row and column dimensions, called axes. Axes are made up of cell-like locations, called nodes. We use the terms axes and nodes, to enrich the spreadsheet paradigm's nomenclature. Each node can have a primary name and any number of alternate names, called tags. Nodes can be indented to form a tree structure, and can be referenced via their path. Path queries can use names and structural keywords (e.g. `Children`, `Siblings`).

Enhanced Referencing Examples

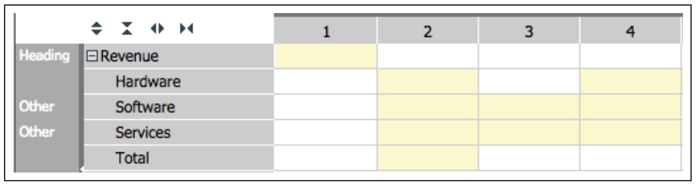
By way of example, the following row references could be used to achieve the selections in Figure 3a.

1) Rows named `Revenue` or tagged `Heading`
2) Children of `Revenue` or `Heading`
3) Rows tagged `Other`
4) Rows that are siblings of `Total`

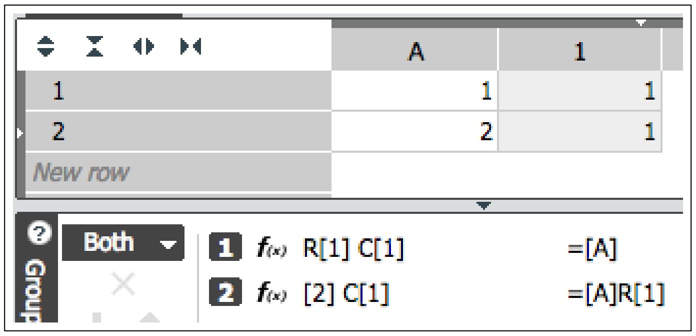
The shortest unambiguous reference can be used. Two examples (see Figure 3b) of this are 1) if only a column is specified, the current row is assumed 2) if a column name does not appear in the row axis, the axis (`C[]` for column) specifier is optional.

Intersections: Ranges are specified as the intersection of rows and columns (e.g. `[Total][Jan:Apr]`). These references have a more natural language feeling definition (as noted in [22]) than the corner to corner ranges in current spreadsheets. This is superior to named ranges [23] as there is no indirection with enhanced referencing, whereas a named range is not much more then an alias for a reference.

(a) Highlighted cells in each column represent a group



(b) Defaults and Disambiguation


	A	1
1	1	1
2	2	1
New row		

Both

1 f(x) R[1] C[1] =[A]

2 f(x) [2] C[1] =[A]R[1]

Fig. 3: Enhanced Referencing Examples

Dynamic Nodes

Dynamic nodes refers to the concept that row and column headers can contain an expression rather than being a literal value. When an expression results in an array value, the axis expands so that from the users' perspective it only contains literals. This is equivalent to a user creating multiple nodes. Cell groupings that include the newly created cells set formulae according to the modeller's intention.

C. Example Applications of Gradual Structuring

Partially Structured Example

In Figure 4 the Scrum Burndown model is refactored. First (fig. 4b) the rows are turned into a semantic axis. Row 1 is renamed to `Tasks` and the three tasks rows are indented under it. Row 5 is renamed `Total`. Next, the column axes are refactored (fig. 4c). Finally (fig. 4d) the day columns in the carry block are replaced with a dynamic node that references the days under `Input`. Two cell groupings (fig. 4e) are created setting the formulae for the cells in the carry and total blocks. The estimate total formula remains unchanged (fig. 2c).

Adding a day in the input block, will automatically get included in the carry block. Cell grouping will ensure that the newly created cells get the prescribed formulae.

Example Summary

The refactoring examples demonstrate the expressiveness of the Gradual Structuring features. Enhanced references, enabled by semantic axes, allow for the creation of cell groupings with one formula applicable to all the cells in a group. These groups remove the copy-paste redundancies, and at the same time are flexible enough to direct changes as the model is modified. Other redundancies, in this case duplicate column headers, are eliminated by imbuing the node of a semantic axis with cell like properties. In fact dynamic nodes are more expressive than cells as they cleverly deal with array values. Unlike pivot tables or spreadsheet cells, these features allow for partial structuring.

V. Discussion

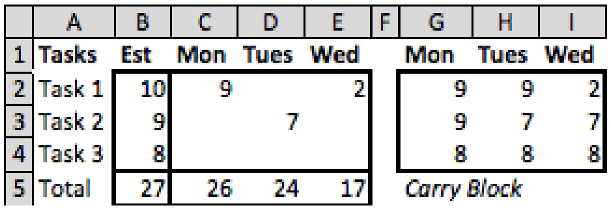
This section considers the consequences of Gradual Structuring from two perspectives; expressiveness and learnability. The conciseness conjecture is used [24] as the definition of expressiveness. In summary

> *"less expressive languages exhibit repeated occurrences of programming patterns, and that this pattern-oriented style is detrimental to the programming process".*

Expressiveness

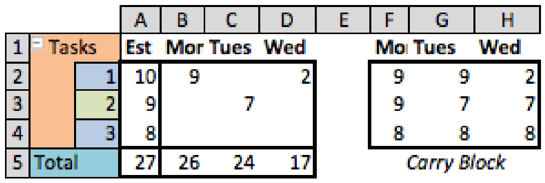
Structured modelling is more expressive, has more abstractions, but is generally less flexible and less usable. Gradual Structuring by definition makes free-form modelling more expressive, as it enables moving between free-form and structured modelling. The examples have shown that the repetition of a formula via copy-paste can be replaced by a cell group and a single reference, and that a single dynamic node can be substituted for a block of repeated headers.

(a) Original Spreadsheet Model



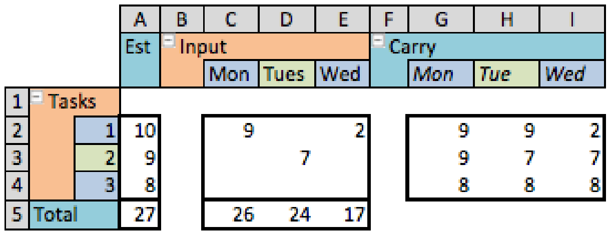
	A	B	C	D	E	F	G	H	I
1	Tasks	Est	Mon	Tues	Wed		Mon	Tues	Wed
2	Task 1	10	9		2		9	9	2
3	Task 2	9		7			9	7	7
4	Task 3	8					8	8	8
5	Total	27	26	24	17		Carry Block		

(b) Refactored Rows - Named and Hierarchically Structured



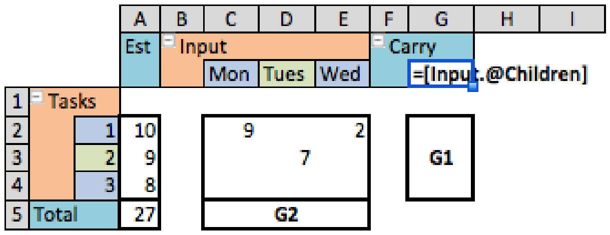
			A	B	C	D	E	F	G	H
1	Tasks		Est	Mor	Tues	Wed		Mo	Tues	Wed
2		1	10	9		2		9	9	2
3		2	9		7			9	7	7
4		3	8					8	8	8
5	Total		27	26	24	17		Carry Block		

(c) Refactoring Columns - Named and Hierarchically Structured



			A	B	C	D	E	F	G	H	I
			Est	Input				Carry			
					Mon	Tues	Wed		Mon	Tue	Wed
1	Tasks										
2		1	10		9		2		9	9	2
3		2	9			7			9	7	7
4		3	8						8	8	8
5	Total		27		26	24	17				

(d) Change Carry Columns to a Dynamic Node



			A	B	C	D	E	F	G	H	I
			Est	Input				Carry			
					Mon	Tues	Wed		=[Input.@Children]		
1	Tasks										
2		1	10		9		2		G1		
3		2	9			7					
4		3	8								
5	Total		27			G2					

(e) Cell Group Definitions, G1 and G2.

```
G1 [Carry.@Children][Tasks.@Children] =
   "=IF(ISBLANK([Input.@This]), ....)"
G2 [Input.@Children][Total] =
   "=Sum([Carry.@This][Tasks.@Children])"
```

Fig. 4: Refactoring Scrum Burndown to a Partially Structured Model

Four language based abstractions have been presented that increase the expressiveness of spreadsheets, with no detrimental effects on liveliness and minimal interference with directness. These features are orthogonal, in that the primary functionality of one cannot be created using any of the others, and when used in combination their behaviour is clear.

None of these features, on there own, are required to perform computation (i.e. be Turing-complete). However the design of these features needs to take into account that power facilitates abuse. For example cell grouping (when considering overlapping cells) or enhanced referencing (conditionals in references) could become Turing-complete. We conjecture that usable expressiveness generally correlates with a decrease in computational power, and goes against the golden hammer rule [25]. Turing-completeness of features does not necessarily need to be avoided, but it should be acknowledged. Overuse of the powerful aspects of features makes a good candidate for code smell detection [26] (e.g. nested IFs).

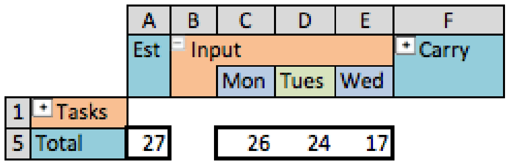
Learnability

This section discusses the learnability of Gradual Structuring from two perspectives. First from a holistic perspective and second from the perspective of the individual features.

Holistic Perspective

Gradual Structuring as a concept should improve learnability as it makes less accessible but more expressive structured modelling available via stepwise refinement. There are a number of routes a learner might take in adopting structured modelling. For instance, as in the worked partially structured example, the rows and columns could be refactored (fig 4d) without using any of the other features. This on its own is of value as a display feature, the nesting enables collapsing and hiding of nested nodes (see Figure 5). Also as a standalone feature enhanced referencing should make formulae easier to read. Similar benefits exist for cell groupings as a standalone feature.

Dynamic nodes possibly has no standalone benefit, as it is strongly dependant on semantic axes and cell grouping before generating any value. Therefore the first three can be considered as anchor concepts [27] for Gradual Structuring.



			A	B	C	D	E	F
			Est	Input				Carry
					Mon	Tues	Wed	
1	Tasks							
5	Total		27		26	24	17	

Fig. 5: Refactoring Model - Collapsed Nodes

Dynamic nodes would be future down the an anchor graph.

Reductionist Perspective

We do not believe there are causal links between features of Gradual Structuring and learnability. Some of the features are naturally intuitive, others potentially counter intuitive or neutral.

Abstractions do not necessarily have learnability penalties, particularly if they are derived to capture common patterns of use [12]. Cell grouping and semantic axes emerged from the design patterns of expert modellers. They are therefore expected to be innately usable features.

However enhanced referencing, does not emerge from pattern capture. There are innate human cognitive abilities that can be used to guide the design of enhanced referencing. For example across all natural languages, subject verb object (SVO) is the most common dominant grammatical form. From [28] (page 373) SOV is the most dominant form (44 percent) across all natural languages, whereas OVS and OSV are not the dominant form in any. We would therefore expect certain enhanced referencing syntaxes to be more usable. For example `[@ChildrenOf Input]` (SVO) should be more usable than `[Input.@Children]` (OVS). Enhanced referencing is therefore a good candidate for empirical studies. Programming languages in general would benefit by investigating the learnability of their features using tools from cognitive psychology.

Dynamic nodes is possibly neutral on learnability. It did not emerge from pattern capture, and is not a modification of an existing feature. It has the ability to provide a directness quality to the use of arrays. Current array formulae are not particularly usable and lack directness. The potential of it becoming a feature with high learnability is promising, but it would need to be taught as it is unlike to be intuitive to most people.

VI. Related work

Structured modelling research often involves basing the spreadsheet paradigm on familiar formal specifications (e.g. calculus [29], object-oriented specification [30], and functional programming [31]). This line of work has a long history [32], [33]. More recently there is research into manipulating and visualizing hierarchical data in spreadsheets [34] and using spreadsheets as an application development platform [35].

Bidirectional transformation of model-driven spreadsheets [36] is an approach to bring structured modelling to spreadsheets. This work is part of the larger SSaaPP project which considered spreadsheet as a programming paradigm [37].

The intersection of spreadsheets and software engineering is an active research area. Of particular note, in relation to this paper is research in refactoring for spreadsheets [38]–[41]. Gradual Structuring would benefit greatly if combined with refactoring.

There is also an overlap between research into spreadsheets and cognitive implications of computational language [42], [43]. Spreadsheets are one of the few programming environments that appear to align with innate human cognitive abilities, as can be inferred from [7]. They therefore hold great promise as a tool for research into usable computational languages. Programming language research should look at the seminal work of Newell and Card [44], which successfully advocated for the application of cognitive psychology to human computer interaction.

Abstractions enabling function decomposition are commonly proposed for the spreadsheet paradigm. A seminal paper in this area is [4]. More recently [45] termed these features as sheet defined functions and has research on the performance of these. One of the issues raised with sheet defined functions is ensuring they have the same power as built-in functions, with first class array values proposed as a possible solution. Dynamic nodes offers another solution, which is possibly easier to integrate and is more usable.

Programming languages are the essential tool for software engineering. There is substantial research into spreadsheets as the most prominent end-user software engineering environment [46]–[48]. It is be beneficial to think of spreadsheets as programming languages [49], and how they can co-evolve with end-user software engineering, in a similar way [50] treats more traditional programming languages.

Gradual Structuring exists to a small extent in current spreadsheet applications. Tables and their associated structured references were added to Excel in the 2007 version. Similar features exist in a Apple Numbers. These are weak forms of Gradual Structuring. This paper advocates that stronger forms should be researched.

VII. Concluding Remarks

Gradual Structuring is the ability to develop models using unstructured and structured modelling features simultaneously. It is important that the same results can be generated using either type of feature. Refactoring can be used to change between them. In an analogy to gradual typing, free-form modelling is programming in a dynamically typed programming language and structured modelling a static typed one.

The features facilitating Gradual Structuring are all abstractions added to the spreadsheet paradigm. A minimal set of requirements for structured modelling are taken from multidimensional modelling applications. A more complete list of four requirements are from OLAP systems.

Modern spreadsheets contain some structured features, most notably pivot tables. However they are not considered to be environments that support Gradual Structuring as it is not possible to refactor between them.

Cell grouping facilitated by enhanced referencing allows more structured models than using copy-paste. Using cell grouping to create fully structured models achieves the 'separation of structure and representation' required from structured modelling.

Enhanced references facilitated by semantic axes are the key technical requirement needed for cell grouping and thereby Gradual Structuring. This is a small feature that can semantically be well defined, but with a lot of syntactic options, as such it is likely to benefit from empirical studies into learnability. Focusing on this area could help generate a design science of programmer experience.

Dynamic nodes solve the problem of header repetition. It suggests the possibility of adding dimensional structuring to the spreadsheet paradigm. This would provide the spreadsheet paradigm with rich dimensional structuring, which is part of the first requirement specified in structured modelling. The hierarchical referencing requirement of structured modelling is provided by path referencing of enhanced references.

Gradual Structuring has the potential to unify the free-form nature of spreadsheets with the expressiveness of structured modelling, whilst retaining liveliness and directness. Liveliness and directness are sought after qualities of programming systems. Making the spreadsheet paradigm simultaneously more expressive, with the same or greater degree of usability.

Spreadsheets can be regarded as a tool for early stage problem formulation. With gradual structuring, early stage modelling can be done in a free form way and structure can be incrementally added as the model matures. In this way Gradual Structuring has the potential to extend the domain of spreadsheets as both a product for expressing complex computation and a learning environment.

References

[1] P. Warren, "Learning to Program: Spreadsheets, Scripting and HCI," in *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30*, ser. ACE '04. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2004, pp. 327–333. [Online]. Available: http://dl.acm.org/citation.cfm?id=979968.980012

[2] J. Cassidy, "The Reinhart and Rogoff controversy: A summing up," *The New Yorker*, no. April, 2013.

[3] R. Panko, "What We Dont Know About Spreadsheet Errors Today," in *EuSpRIG*, 2015.

[4] S. P. Jones, A. Blackwell, and M. Burnett, "A user-centred approach to functions in Excel," in *ACM SIGPLAN Notices*, vol. 38, no. 9. ACM, 2003, pp. 165–176. [Online]. Available: http://dl.acm.org/citation.cfm?id=944721

[5] R. Samurcay, "The concept of variable in programming: Its meaning and use in problem-solving by novice programmers," *Studying the novice programmer*, vol. 9, pp. 161–178, 1989.

[6] W. V. O. Quine, "Word & Object," 1960.

[7] A. Ambler, "Generalizing the sheet language paradigm," in *Visual Languages and Applications*, 1990.

[8] A. Stefik and S. Hanenberg, "The programming language wars: Questions and responsibilities for the programming language community," in *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*. ACM, 2014, pp. 283–299.

[9] A. Takikawa, D. Feltey, B. Greenman, M. S. New, J. Vitek, and M. Felleisen, "Is sound gradual typing dead?" in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, 2016, pp. 456–468.

[10] Facebook, "Hack." [Online]. Available: http://hacklang.org/

[11] Google, "Closure Compiler." [Online]. Available: https://developers.google.com/closure/compiler/

[12] E. Visser, "Understanding Software through Linguistic Abstraction," *Science of Computer Programming*, 2013.

[13] J. Sajaniemi, "Modeling spreadsheet audit: A rigorous approach to automatic visualization," *Journal of Visual Languages & Computing*, vol. 11, no. 1, pp. 49–82, 2000.

[14] F. Hermans and T. V. D. Storm, "Copy-Paste Tracking : Fixing Spreadsheets Without Breaking Them Copy-Paste Tracking in Action," 2013.

[15] J. F. Raffensperger, "The Art of the Spreadsheet - Organize blocks with care." 2008. [Online]. Available: http://john.raffensperger.org/ArtOfTheSpreadsheet/ Chapter05_BeConciseWithBlocks.html

[16] S.-C. Cheung, W. Chen, Y. Liu, and C. Xu, "CUSTODES: Automatic Spreadsheet Cell Clustering and Smell Detection using Strong and Weak Features," 2016.

[17] "ModelOff Questions." [Online]. Available: http://www.modeloff.com/questions/

[18] E. Thomsen, *OLAP solutions: building multidimensional information systems*. John Wiley & Sons, 2002.

[19] Lotus, "Lotus Improv." [Online]. Available: https://en.wikipedia.org/wiki/Lotus_Improv

[20] Quantrix, "Quantrix® Modeler Version 4.0 User Guide." [Online]. Available: https://www.quantrix.com/quantrix/userfiles/file/quantrix modeler user guide.pdf

[21] D. Miller, G. Miller, and L. M. Parrondo, "Sumwise : A Smarter Spreadsheet," in *EuSpRiG*, 2010. [Online]. Available: http://www.sumwise.com/wordpress/wp-content/uploads/Sumwise-A-Smarter-Spreadsheet.pdf

[22] E. Aivaloglou, D. Hoepelman, and F. Hermans, "A Grammar for Spreadsheet Formulas Evaluated on Two Large Datasets," 2015.

[23] R. McKeever and K. McDaid, "How do range names hinder novice spreadsheet debugging performance?" in *EuSpRIG*, 2010.

[24] M. Felleisen, "On the expressive power of programming languages," in *ESOP'90*. Springer, 1990, pp. 134–151.

[25] W. H. Brown, R. C. Malveau, and T. J. Mowbray, "AntiPatterns: refactoring software, architectures, and projects in crisis," 1998.

[26] F. Hermans, M. Pinzger, and A. V. Deursen, "Detecting Code Smells in Spreadsheet Formulas," *Proceedings of the International Conference on Software Maintenance (ICSM)*, 2012.

[27] J. Mead, S. Gray, J. Hamer, R. James, J. Sorva, C. S. Clair, and L. Thomas, "A Cognitive Approach to Identifying Measurable Milestones for Programming Skill Acquisition," in *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '06. New York, NY, USA: ACM, 2006, pp. 182–194. [Online]. Available: http://doi.acm.org/10.1145/1189215.1189185

[28] J. R. Anderson, *Cognitive psychology and its implications* ., fifth edit ed. WH Freeman/Times Books/Henry Holt & Co, 2000.

[29] M. Erwig, R. Abraham, S. Kollmansberger, and I. Cooperstein, "Gencel: a program generator for correct spreadsheets," *Journal of Functional Programming*, vol. 16, no. 03, p. 293, 2006.

[30] G. Engels and M. Erwig, "ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 2005, pp. 124–133.

[31] F. o. I. Csernoch, Maria (University of Debrecen and F. o. I. Biró, Piroska (University of Debrecen, "Sprego Programming," *Spreadsheets in Education (eJSiE)*, vol. 8, no. 1, 2015.

[32] A. Ambler, "Forms: Expanding the visualness of sheet languages," *1987 Workshop on Visual Languages*, pp. 105–117, 1987.

[33] K. Hassinen, J. Sajaniemi, and J. Vaisanan, "Structured spreadsheet calculation," *[Proceedings] 1988 IEEE Workshop on Languages for Automation@m_Symbiotic and Intelligent Robotics*, 1988.

[34] K. S.-P. Chang and B. A. Myers, "Using and Exploring Hierarchical Data in Spreadsheets," *ACM CHI*, 2016.

[35] R. M. Mccutchen, S. Itzhaky, D. Jackson, R. M. Mccutchen, and S. Itzhaky, "Initial report on Object Spreadsheets," 2016.

[36] J. Cunha, J. P. Fernandes, J. Mendes, H. Pacheco, and J. Saraiva, "Bidirectional Transformation of Model-Driven Spreadsheets." *ICMT*, vol. 12, pp. 105–120, 2012.

[37] R. Abreu, T. Alves, O. Belo, J. C. Campos, J. P. Fernandes, P. Martins, J. Mendes, H. Pacheco, C. Peixoto, R. Pereira, H. Ribeiro, A. Riboira, J. Saraiva, J. C. Silva, and J. Visser, "SSaaPP: SpreadSheets as a Programming Paradigm," Tech. Rep. August, 2014.

[38] S. Badame and D. Dig, "Refactoring meets spreadsheet formulas," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 399–409.

[39] J. Cunha, J. Mendes, J. Saraiva, and J. Visser, "Model-based programming environments for spreadsheets," *Science of Computer Programming*, vol. 96, pp. 254–275, 2014.

[40] D. J. Hoepelman, "Tool-assisted Spreadsheet Refactoring and Parsing Spreadsheet Formulas," Ph.D. dissertation, TU Delft, Delft University of Technology, 2015.

[41] F. Hermans, E. Aivaloglou, and B. Jansen, "Detecting problematic lookup functions in spreadsheets," in *Visual Languages and Human-*

Centric Computing (VL/HCC), 2015 IEEE Symposium on. IEEE, 2015, pp. 153–157.

[42] B. Kankuzi and J. Sajaniemi, “A mental model perspective for tool development and paradigm shift in spreadsheets,” *International Journal of Human-Computer Studies*, vol. 86, pp. 149–163, 2015.

[43] P. Saariluoma and J. Sajaniemi, “Visual information chunking in spreadsheet calculation,” *International Journal of Man-Machine Studies*, vol. 30, no. 5, pp. 475–488, 1989.

[44] A. Newell and S. Card, “The Prospects for Psychological Science in Human-Computer Interaction,” *Human-Computer Interaction*, vol. 1, no. 3, pp. 209–242, 1985.

[45] P. Sestoft, J. D. Rask, and S. Eikeland, “End-user development via sheet-defined functions,” *Software Engineering Methods in Spreadsheets*, p. 8, 2014.

[46] A. J. Ko, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, S. Wiedenbeck, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, and H. Lieberman, “The state of the art in end-user software engineering,” *ACM Computing Surveys*, vol. 43, no. 3, pp. 1–44, 2011.

[47] R. Abraham, “End-User Software Engineering in the Spreadsheet Paradigm,” Ph.D. dissertation, 2007.

[48] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva, “Spreadsheet engineering,” in *Central European Functional Programming School*. Springer, 2015, pp. 246–299.

[49] F. Hermans, “Analyzing and Visualizing Spreadsheets,” Ph.D. dissertation, 2012.

[50] E. Murphy-Hill and D. Grossman, “How Programming Languages Will Co-evolve with Software Engineering: A Bright Decade Ahead,” in *Proceedings of the on Future of Software Engineering*, ser. FOSE 2014. New York, NY, USA: ACM, 2014, pp. 145–154. [Online]. Available: http://doi.acm.org/10.1145/2593882.2593898