Decision-making in software development

Tom McBride
University of Technology, Sydney)
mcbride@it.uts.edu.au

Abstract

There are many decisions made during the processes of software development and there are several decision-making methods that could be used in any specific circumstance. International standards for software and systems engineering tend to assume a specific decision-making method will be used in some processes while in others the decision-making method is implied rather than explicitly stated. An empirical study investigated which decision-making methods were used by practicing software developers. The study found that practitioners used a variety of methods, highlighting the need for flexible processes and flexible assessment of those processes where decision-making is concerned

1. The problem.

Many decisions are made during the processes of software development and while the main methods of decision-making are reasonably well known (see e.g. [1, 2]), there has been little investigation into which methods are used or appropriate for the different software development decisions. Decision-making falls into two philosophical types: rational and naturalistic [1]. Rational decision-making generally proceeds by establishing goals, seeking information to understand the situation and the alternative solutions then choosing from among the alternatives. Naturalistic decision-making [3] does not seek several alternatives but simply implements the first workable alternative. The different circumstances that favour one or other are shown in
The paper proceeds by discussing the methods and models of decision-making then examining the decision-making in software development from a theoretical perspective. It then describes the models of decision-making in the two main international standards, ISO 12207 and ISO 15288, before proposing a research question. The research method and analysis is presented. Threats to validity are briefly discussed before presenting the findings, discussion and conclusion.
The origins of rational choice can be traced back to Bernoulli [4] with variations and refinements being added in more recent times, notably by Simon [2]. The essential characteristics of rational choice are that several alternatives are investigated before choosing the best. Analysis of alternatives is often performed using specific techniques such as cost-benefit analysis or, in the case of software architecture, architecture trade-off analysis method [5].
Rational choice is expensive to perform compared to alternative methods such as naturalistic decision-making, but it is well suited to big, complex decisions. This is most likely to be encountered in large, complex software development projects such as military, government or aerospace.
For software development projects that are not large, complex or critical, requiring that decision-making be performed using rational choice may incur cost without additional benefit and alternative methods may be more appropriate. There is also a lot of anecdotal evidence that software architectures, in particular, are frequently based on the previous version or a similar system rather than developed from a set of alternatives. Unfortunately, software and system engineering life cycle processes described in international standards, such as software architecture design [6], assume specific decision-making methods will be used, possibly unintentionally.
If software development in the field uses different decision-making approaches in different circumstances then the rigidity of international standards is likely to incur cost without benefit and cause unnecessary difficulty for practitioners.

Table 1: Boundary conditions for different decision strategies from [3]

Task conditions	Naturalistic	Rational
Greater time pressure	More likely	
Higher experience level	More likely	
Dynamic conditions	More likely	
Ill-defined goals	More likely	
Need for justification		More likely
Conflict resolution		More likely
Optimization		More likely
Greater computational complexity		More likely

To date there has been little empirical investigation into which decision-making methods are used by software and system developers for the different decisions they must make during software development. This paper describes a small empirical investigation and reports its findings. The paper proceeds by discussing the methods and models of decision-making then examining the decision-making in software development from a theoretical perspective. It then describes the models of decision-making in the two main international standards, ISO 12207 and ISO 15288, before proposing a research question. The research method and analysis is presented. Threats to validity are briefly discussed before presenting the findings, discussion and conclusion.

2. Models and methods of decision-making

Naturalistic decision-making was identified by Klein [3] as a consequence of a study into how people make decisions under time pressure. The motivation for the study was to improve decision-making in combat conditions and was conducted, not in combat, but by studying how fire fighters make decisions.

In such circumstances the consequences of the decisions are high, information is usually incomplete and often subjective, knowledge of the probable outcomes is limited and the decisions are made by experienced people. Naturalistic decision-making, as described by Klein, relies on the decision-maker's experience that is gained through many observations and real-life case studies.

Although well removed from the life-threatening nature of combat or fires, decisions made by software developers during the early phases of a project have many of the same characteristics. The requirements are often uncertain, decisions can not be delayed until many of the uncertainties are resolved, the technology is constantly changing and there is considerable pressure to "do something". Here experience can also play a major role.

Klein's naturalistic decision-making method is not the only method proposed for time-critical situations. Also originating from the military is Boyd's observe-orient-decide-act (OODA) model and several others are reviewed by Azuma *et al* [1].

Rational choice decision-making, by contrast, places great emphasis on information gathering, alternative generation and selection from among the alternatives. Decisions concerning whether to build a replacement nuclear reactor or the distribution of water rights in drought prone areas have significant consequences, will need to be justified, have many stakeholders and there are few precedents. Many techniques of decision analysis have emerged to assist with decisions in different domains from multi attribute analysis, decision structuring, probabilistic risk analysis through to Bayesian Belief Networks [7].

The circumstances of decision-making seldom fall neatly into those of either rational choice or naturalistic, so there is frequently an element of ambiguity over which decision-making approach is better suited to the circumstances.

3. Decision-making in software development

While there are many decisions made during software development, some seem to be representative of different types of decision or, at least, different circumstances that may call for different decision-making. There are always decisions about when some phase of work is sufficiently complete to proceed to the next phase. An example of this would be deciding whether that the requirements were sufficiently established to allow work to proceed on the basic architecture, or whether the architecture had achieved a standard sufficient to allow the next phase to begin. One of the final decisions of software development is deciding that the software is fit to be released to the end users or customers. In this type of decision there is normally opportunity to seek sufficient information to make an informed decision.

A different type of decision concerns software architecture. Gasson [8] argues that software design has some of the characteristics of a "wicked problem" [9] that cannot be stated or solved in any real sense but, instead, are socially constructed and subjective. An architecture is not right or wrong so much as "more or less fit for some stated purpose" [5 p15]. That software design is a creative act is not universally agreed. While acknowledging the role and need for creativity Gordon [10] argues that design should utilise functional decomposition, implying that sufficient decomposition will overcome most design challenges.

Quite a different decision is that concerning which of several candidate defects should be attended to next, or which of several change requests should be processed next. Such decisions are amenable to being expressed in a procedure, requiring that there be some method to manipulate the relevant attributes of the defects or change requests to identify which of them should have priority. There are several such decisions in software development with the common characteristic that the decision must be made frequently so that criteria can be developed to guide the decision-making for most circumstances.

4. Decision-making in ISO 12207 and ISO 15288

The two main international standards covering the field of software and systems engineering are ISO 12207 [6] and ISO 15288 [11] respectively. Both of these standards have been revised and have been submitted for their final ballot to be published as official standards, replacing their earlier versions. This paper examines the decision-making within the processes of these final draft international standards.

Decision-making in ISO 12207 and ISO 15288 falls into five categories;

1. Processes whose sole purpose is to make one or more decisions. Software Qualification Testing Process has a purpose of the one decision - that the integrated software product meets its requirements.

2. Explicit decision-making as one of the activities of the process. System Architectural Design and Software Architecture Design Processes contain an activity to "analyse and evaluate the architecture" in which the tasks describe a decision-making process.

3. Significant decisions that are implied by the process. Requirements Analysis Process contains the implicit task of deciding whether or not to include requirements in the system to be developed but this is not recorded as a process outcome or activity.

4. A negotiated decision. The Acquisition Process contains a task of negotiating a contract between supplier and acquirer. While it may seem that negotiation and decision-making are different processes, negotiation can also be regarded as a means to arrive at a decision, in contrast to evaluation against specified criteria.

5. Procedural decision. Some processes, such as Risk Management or Software Problem Resolution, describe a full cycle of information gathering, evaluation against specified criteria and actions taken based on the evaluation results. In other words, a decision-making process in a specific context that has many of the attributes of a procedure. This differs from the normal characteristics of rational decision-making in that no set of alternatives are generated and there is no choice about actions to be taken.

Both standards include a specific Decision Management process that tries to avoid pre-determining the decision-making method but requires that decisions "evaluate the consequences of alternative actions" thus assuming that all decision-making will be rationalistic.

5. Research question

Different circumstances demand different decision-making methods yet the software and system development processes described in two international standards assume that all decisions will be made rationally using specified criteria to evaluate the information and guide the decision or to evaluate the decision outcome.

Developing and specifying criteria to be used in decision-making, especially when the decision-making must be transparent and traceable, involves some cost to the organization in terms of time and resources. Some organizations, especially very small enterprises, and some circumstances need not necessarily require such decision-making rigour. To date there has not been any empirical investigation of how software development organizations make various decisions. It would seem that both the software and systems development community and the standards making community could benefit from an investigation into "How do software development organizations make decisions?"

6. Research method

Research participants were selected opportunistically through professional contacts, meetings at industry presentations and other ways that resulted in finding a software developer prepared to be interviewed about their, or their organization's, software design and decision processes.

Interviews were semi structured without a fixed list of questions. Each interview proceeded from questions concerning how they designed their software applications and how they decided that a design was acceptable. The interviews sought information on decision-making about feature sets, decision-making about software design, decision-making on releasing the software to the customer or end user and decision-making on defect prioritization and fixing. All interviews were recorded, with the participant's permission, using a digital note-taker. The interviews were not transcribed but were summarised to capture the essential points made during the interview, then emailed back to the participant for their comment and correction. After any corrections had been received, the summary and the audio file were encoded into a standard template for analysis (Table 2).

The main topic of the survey was decision-making in software architecture and the interview began by asking respondents how they, or their organization, decided on the system architecture or decided that the architecture was "good". When that had been discussed through exploration of the different phases of decision-making, the interview turned to other decisions such as how the organization decided which defects to fix next or decided on the composition of a feature set. Asking about two different decisions enabled a contrast between different decision-making methods within the one organization.

Table 2: Example template for analysis showing example data

Decision phase	Architecture - Design of a new or changed feature.
Sense-making	Deep Extensive knowledge of the product. Expressed "gut feel", 28 years experience.
Information search	Future "What would be nice to have" The more information you can get, the better. Customers are the worst – so concerned with their day-to-day jobs. Support people – listen to customers "little whinges

	(complaints)" Some of the more technical savvy managers. ACM, IEEE magazines. Competitive products, white papers, web sites.
Decision trigger	Anticipate need for something. Prototype a possible answer. Official trigger is a new or changed feature.
Decision inputs Requirements / constraints	Things I think would be nice to do. Feature overflow. Can't do them now, but can think about the hooks that would be needed to ease their future adoption.
System qualities	Assumed.
Existing/legacy system	Design rules in the original product puts very few restrictions on possible solutions. Very few of them and now "folk lore" Try to have a few constraints as possible. Knowledge of the facilities available in the existing systems. Knowledge of what the future product should be, how it should behave.
Political constraints	Not mentioned.
Decision process	Naturalistic Conceive an idea then prototype it. No discussion of the decision-making process itself. Very much the engineer viewpoint that something is "obvious". Extensive knowledge of the product, its design and the working environment means that some options are not even considered and others are "obvious" to the expert. So, expert judgment.
Decision Criteria Quantitative/ Qualitative/ Impressionistic	Prototype first to test the idea out. Will it work? Will it solve the problem? Does it retain the internal qualities that we want in the product.

7. Analysis

Twenty participants from twelve different organizations gave information about thirty seven decisions. The majority of those decisions concerned the software architecture or design (17), with the next largest category being product release decisions (8) followed by defect management decisions (6), feature set decisions (3) with process, product enhancement and technology adoption at one each. It should not be surprising that decisions about software architecture or design were the most populous because that was the main purpose of the interview.

A rational decision-making process was judged to have been used in seventeen decisions, a procedural process on eleven decisions, a naturalistic decision-making process on five decisions and on four decisions the decision-making process is best described as negotiated.

A procedural decision is made through following a preset procedure with preset criteria directing the decision. For example, the decision on which defects to investigate and correct first tends to be made based on the assessed importance of the defect and is often simply a matter of sorting the collected defects by the various categories such as priority, impact, estimated cost to fix, etc., then accepting the results of the sort. Similarly, a decision to release a product for use by end users or customers tends to be made by setting criteria for release well prior to the decision then releasing the software as soon as it has achieved the set criteria. Such decisions do not require expert judgment.

A negotiated decision is where a decision is subject to modification through a sequence of negotiations among the interested parties. For example, on respondent said that many of their design decisions did not depend on the expertise of the designers so much as the quality of the reviews of those designs. The outcome of the reviews, and there were usually more than one, was usually to require some modification to the design so that it didn't matter so much what the original design was because the reviews would "negotiate" the design into acceptability.

Decisions about architecture or design were mostly made rationally through consideration of alternatives and not naturalistically. Rather than relying on past experience and intuition, software architects consider a range of alternatives, usually patterns but sometimes designs, for the major components before making a decision. The architecture usually proceeds stepwise through levels of decomposition. Of interest were those who claimed that the architecture was arrived at through negotiation rather than rational choice. Here the architects proposed design solutions for components of the system and the decision was made through negotiations among the separate architects. More than one respondent said that their system architectures were based on previous, existing systems. They would start by asking "what is this system like?" and search for a system that solved a similar problem then adapted it to the current situation.

Selecting defects to fix, deciding on the composition of a feature set and deciding when the software was ready to be released were all decided mainly by procedure. That is, there were established steps to be used to gather and analyse the required information and specific criteria that guided the decision. For example, almost everyone classified defects in some way that allowed them to be

prioritised. With that done, the highest priority defects were fixed first. Similarly, feature sets were decided based on the priority of the requirements, change requests and defect fixes. In almost all cases respondents gave the impression that they were surprised to be asked about such matters, that these were so routine as to be a widely accepted way of doing things rather than something requiring a "decision".
The three unique decisions about technology adoption, process tailoring and product enhancement could not be classified easily into the other categories and are included for completeness. A summary of these findings is presented in Table 3.

Table 3: Summary of findings

	Decision-making method			
Decision	Rational	Naturalistic	Negotiated	Procedural
Feature set (which requirements, enhancements and defect fixes are included in the next release of the software)	1			2
Architecture/Design (How was the architecture decided?)	10	4	3	0
Release (Is the software fit for release to the user)		2		6
Defect (which defects should be investigated and fixed)	1	2		3
Technology adoption (choice between several alternative technologies)	1			
Product enhancement (what features should be developed for the next release)	1			
Process (how should the development process be tailored)			1	

8. Threats to validity

The number of participants in this research less than would be necessary to achieve statistical validity. Also all of the participants were located in Sydney, Australia where software projects are seldom large enough to require a team of more than 100 software developers. This bias toward smaller projects limits the validity of applying any findings to larger projects, say 1000 developers.
This analysis relied on the experience of the researcher to interpret the data consistently. There was no independent check of the consistency. However, the researcher is an experienced assessor as well as an academic accustomed to the demands of marking assignments so claims that the habits of consistency have been established.
The use of a digital notetaker allowed the recordings themselves to be retained thereby enabling the analysis and findings to be checked by an independent party. However, removing interview transcripts and relying on the original recordings is relatively new and its validity as a research method is yet to be tested and accepted.

9. Discussion

The findings clearly show that there is seldom universal agreement on a decision-making method, which is unsurprising. Large, complex problems require a different decision-making method from large simple problems. In software architecture, for example, the architecture of the next generation of a product is unlikely to be significantly different to its predecessor, even if the implementation technology is different. As one interviewee remarked

> *"Most business problems are not unique. Most business processes are similar to something else so we start somewhere. It's a matter of asking "What's it like?" You start organising things into objects that make sense according to similar systems."*

With the introduction of design patterns [12] software development gained a method of expressing design abstractions that enabled the essence of a design to be communicated. What was once tacit knowledge gained through years of experience could now be taught. More design problems became amenable to deterministic solution, a matter of finding the appropriate combination of design patterns. That said, there are some systems and some problems that are sufficiently unique that relying on precedent solutions is not viable.
Decisions about feature sets, readiness for release and which defects to fix also are not made by everyone using the same method.
While it may be easy to argue that the choice of decision-making method is individual, thus arguing for the interchangability of decision-making methods, it is also possible that there are other factors at play. This research has so far not investigated any links between the size of the problem and decision-making method. For example, if a person or organization had only a small number of defects to manage then they may well not bother with formally classifying the attributes of each defect before deciding which to fix. Instead they are more likely to decide informally using nothing other than their personal knowledge of the defects. Similarly risk management might not require formal risk classification and tracking if

there are a small number of risks to manage, although it is more difficult to be sure that the number of risks is what small unless they are enumerated in some fashion.

10. Conclusions and further research

Decision-making in software development is performed using a variety of decision-making methods, even for the same decision. It follows that standard software development methodologies and international software development processes should avoid specifying or assuming a specific decision-making method and should, instead, allow a range of methods to be used. For related activities such as process assessment it also follows that the assessment methods should avoid requiring evidence that is the specific product of a specific decision-making method and, instead, require evidence that could be produced by a range of decision-making methods. Further research is required to investigate which circumstances favour specific decision-making methods so that better guidance can be given about tailoring software development processes and assessment requirements.

11. References

[1] R. Azuma, M. Daily, and C. Furmanski, "A review of time critical decision making models and human cognitive processes," 2006, p. 9 pp.

[2] R. Lipshitz, G. Klein, J. Orasanu, and E. Salas, "Taking stock of naturalistic decision making," *Journal of Behavioral Decision Making*, vol. 14, p. 331, Dec 2001 2001.

[3] G. Klein, *Sources of Power: How people make decisions*. Boston: MIT Press, 1998.

[4] D. Bernoulli, "Exposition of a New Theory on the Measurement of Risk " *Econometrica (Translation from the 1738 original),* vol. 22, p. 14, Jan 1954.

[5] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, 2nd ed.: Addison-Wesley, 2003.

[6] International Standards Organization, ISO FDIS 12207:2007 - Systems and software engineering - software life cycle processes

[7] W. Edwards, J. Ralph F. Miles, and Detlof von Winterfield, "Advances in Decision Analysis: From Foundations to Applications," W. Edwards, J. Ralph F. Miles, and Detlof von Winterfield, Eds.: Cambridge University Press, 2007.

[8] S. Gasson, "Framing design: a social process view of information system development," in *International Conference on Information Systems*, Helsinki, 1998, pp. 224-236.

[9] H. W. J. Rittel and M. M. Webber, "Dilemmas in a General Planning Theory," *Policy Sciences*, vol. 4, pp. 155-169, 1973.

[10] A. M. Gordon, "Software design," *Design Studies*, vol. 5, pp. 68-72, 1984.

[11] International Standards Organization, ISO 15288:2002 - Systems engineering - System life cycle processes

[12] E. Gamma, R. Helm, R. Hohnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley Publishing Company, 1995.