What makes software design effective?

Antony Tang and *Aldeida Aleti,* Faculty of ICT, Swinburne University
of Technology, Hawthorn, Vic 3122, Australia
Janet Burge, Computer Science and Software Engineering, Miami University,
Oxford, OH 45056, USA
Hans van Vliet, Department of Computer Science, VU University
Amsterdam, 1081 HV Amsterdam, The Netherlands

*Software design is a complex cognitive process in which decision making plays
a major role, but our understanding of how decisions are made is limited,
especially with regards to reasoning with design problems and formulation of
design solutions. In this research, we have observed software designers at work
and have analysed how they make decisions during design. We report on how
factors such as design planning, design context switching, problem-solution
co-evolution and the application of reasoning techniques influence software
design effectiveness.*

Software design is a highly complex and demanding activity. Software
designers often work in a volatile environment in which both the business requirements and the technologies can change. Designers also face
new problem domains where the knowledge about a design cannot be found
readily. The characteristics and behaviors of the software and the hardware
systems are often unknown and the complexity in user and quality requirements are high. Under such a complex environment, the ways design decisions
are made can greatly influence the outcome of design. Despite such challenges,
we understand little about how software designers make design decisions, and
how and if they reason with their decisions. We also understand little between
the decision making process and how it influences the effectiveness of the design process.

The University of California, Irvine, prepared an experiment in which software designers were given a set of requirements to design a traffic simulator.
Their design activities were video recorded and transcribed. This gave us the
opportunity to study and compare design reasoning and decision making by
professional designers working on the same problem. The primary goal of
our study is to investigate software design from a decision making perspective.
In particular, we want to understand software design decision making in terms

Corresponding author:
Antony Tang
atang@swin.edu.au

ELSEVIER

614

of how software designers use reasoning techniques when making decisions. If different reasoning and decision making approaches are used, would they influence software design effectiveness?

In this study, we analyse designer activities from three perspectives:

- *Design planning*. Understand the ways software designers plan and conduct a design session and how they influence design effectiveness.
- *Design problem-solution co-evolution*. Understand the ways designers arrange their design activities during the exploration of the problem space and the solution space. For instance, we want to understand if designers identify design problems before formulating solutions; and what would be the implications on design effectiveness.
- *Design reasoning*. Understand the ways designers reason with their design decisions. For instance, would designers be more effective if they create more design options, or if they are explicit about why they make their design choices.

We postulate that these perspectives in combination influence the effectiveness of design activities. Design effectiveness is defined in terms of requirement coverage and design time utilisation. That is, given a certain amount of design time, how can one team address more requirements? Does one team solve a design problem more quickly than another team because of their design decision making activities? In this study, we carry out protocol encodings of the transcripts. We label each part of the designer conversations to identify the decision making and reasoning activities that went on during the design. The encoded protocols are analysed to create visual decision maps. We evaluate the results to investigate the factors that influence the effectiveness of software design. We have found that the ways decisions are made impact on the effective use of design time and consequently the derived solutions.

1 Prior works in design decision making and reasoning

1.1 Design planning

Given a set of software requirements, designers must look at how to structure or plan a design approach. Zannier et al. have found that designers use both rational and naturalistic decision making tactics (Zannier, Chiasson, & Maurer, 2007). Designers make more use of a rational approach when the design problem is well structured, and inversely designers use a naturalistic approach when the problem is not well structured. Goel and Pirolli reported that on average 24% of statements made by the architects are devoted to problem structuring and this activity occurs mostly at the beginning of the design task (Goel & Pirolli, 1992). Christiaans reported that 'the more time a subject spent in defining and understanding the problem, and consequently using their own frame of reference in forming conceptual structures, the better able he/she

was to achieve a creative result.' (Dorst & Cross, 2001). These studies show that design structuring or planning is an important aspect of design.

1.2 Design reasoning

Designers are mostly not cognisant of the process of design decision making. Despite lacking the knowledge about how design decisions are made, which is fundamental to explaining the cognitive process of design, software practitioners and software engineering researchers continue to invent development processes, modeling techniques and first principles to create software products. In real-life, it seems that most of the software designers make design decisions based on personal preferences and habits, applying some forms of reasoning and ending up with varying design productivity and quality.

This phenomenon can perhaps be explained by the way humans think. Researchers in psychology suggest that there are two distinct cognitive systems underlying reasoning: System 1 comprises a set of autonomous subsystems that react to situations automatically, they enable us to make quicker decisions with a lesser load on our cognitive reasoning. System 1 thinking can introduce belief-biased decisions based on intuitions from past experiences; these decisions require little reflection. System 2 is a logical system that employs abstract reasoning and hypothetical thinking, such a system requires longer decision time and it requires searching through memories. System 2 permits hypothetical and deductive thinking (Evans, 1984, 2003). Under this dual process theory, designers are said to use both systems. It seems that, however, designers rely heavily on prior beliefs and intuition rather than logical reasoning, creating non-rational decisions. Furthermore, the comprehension of an issue also dictates how people make decisions and rational choices (Tversky & Kahneman, 1986). The comprehension of design issues depends on, at least partly, how designers frame or structure the design problems. If the way we think can influence the way we design and eventually the end results of the design, then it is beneficial to investigate how software designers make design decisions.

Rittel and Webber (1973) view design as a process of negotiation and deliberation. They suggest that design is a 'wicked problem' in which it does not have a well-defined set of potential solutions. Even though the act of design is a logical process, it is subject to how a designer handles this wicked problem. Goldschmidt and Weil describe design reasoning as the relationship between contents and structure (Goldschmidt & Weil, 1998). They suggest that the process of reasoning as represented by design moves is 'double speared' where one direction is to move forward with new design, another direction is to look backwards for consistency.

As software design experience can be application domain dependent, it is different from other engineering design disciplines where the context of the domain is relatively constant. For instance, the issues faced by software

designers working in the scientific domain are quite different to those working in the transactional financial system domain, so a software designer may use different decision making processes if s/he is faced with unfamiliar domains and technologies. Cross suggests (Cross, 2004) that expert designers appear not to generate a wide range of alternatives. We have found in an earlier study that experienced software designers intuitively rule out unviable design alternatives whereas inexperienced software designers can benefit from explicitly considering design options (Tang, Tran, Han, & Van Vliet, 2008).

1.3 Design problem-solution co-evolution

Although many have researched software engineering design, no studies address the cognitive aspects of the software designers that we are aware of. The studies that are closest to this subject address the use of design rationale in software engineering, mostly from the perspective of documenting design decisions instead of a systematic approach to applying software design reasoning (Curtis, Krasner, & Iscoe, 1988; Dutoit, Mccall, Mistrik, & Paech, 2006; Tang, Barbar, Gorton, & Han, 2006; Tyree & Akerman, 2005). From the problem solving perspective, Hall et al. suggest that problem frames provide a means of analysing and decomposing problems, enabling the designer to design by iteratively moving between the problem structure and the solution structure (Hall, Jackson, Laney, Nuseibeh, & Rapanotti, 2002). These software design methodologies employ a blackbox approach that emphasises the development process and its resulting artifact with little exploration of the cognitive and psychological aspects of designers who use them. Such methods have a resounding similarity with the Mahler model of co-evolution between the problem and solution spaces (Maher, Poon, & Boulanger, 1996), which is also described by Dorst and Cross in (Dorst & Cross, 2001). They suggest that creative design is developing and refining together both the formulation of a problem space and ideas in the solution space (see Figure 1).

Simon and Newell suggest six sources of information that can be used to construct a problem space (Simon & Newell, 1972), providing a context or a task environment for the problem space. Defining the context is thus an important aspect to formulating and identifying a problem. Similarly, we suggest that decisions are driven by decision inputs, or the context of a design problem, and the results of a decision are some decision outcomes. The AREL design rationale model (Tang, Han, & Vasa, 2009) represents the causal relationships between design concerns, design decision and design outcomes. The concerns and the requirements are the context of a design problem. A design problem needs to be analysed and formulated. When a decision is made to select a solution, certain evaluation of the pros and cons of each alternative takes place. A design outcome represents the chosen design solutions. Often one design decision leads to another design decision, and the outcome of a decision can become the design input of the next decision. In this way, a design is a series of decision problems and solutions (Hofmeister et al., 2007; Tang et al., 2009).

Figure 1 Problem-solution co-evolution.
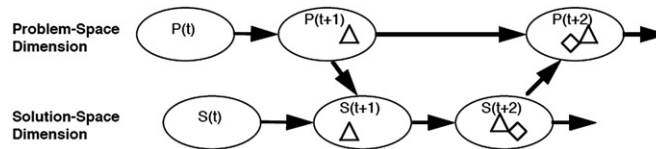
2 Studying software design reasoning and decision making

2.1 The design assignment

The design task set out in the experiment for the software designers is to build a traffic simulation program. The system is meant to be used by students to help them understand the relationships between traffic lights, traffic conditions and traffic flow. The context of the subject is about traffic lights and intersections which is common knowledge that designers should be familiar with. This general knowledge allows the designers to begin their work. However, the building of the simulation software requires specialized and detailed domain knowledge that general software designers are unlikely to possess, so none of the designers have any such advantage over the others that could bias the results. The experiments involved three design teams, each consisted of two designers. Each team was given 2 h to produce a design. We have selected to study two teams only: Adobe Systems Incorporated (Adobe in short) and AmberPoint, we call them Team A and Team M respectively. Both team A and M took the full 2 h to design and a third team (Anonymous) took just under 1 h and did not design the system fully. We chose to study teams A and M only because we cannot compare the effectiveness of the Anonymous team due to the short time they had spent on their design.

The designers were given a Design Prompt or a brief specification of the problem where there are twenty explicit functional and non-functional requirements, and fifteen derived requirements (see Table 1). Derived requirements are requirements that are not specified in the Design Prompt but need to be addressed to complete the design. They are derived from the analysis by the designers plus the authors' interpretation of the requirements[1]. For instance, connecting intersections in the simulation program is a derived requirement.

Table 1 Number of requirements in the system.

Requirements	Network of Roads	Intersection/ Light	Simulate Traffic Flows	Traffic Density	Non-Functional Requirements
Explicit	4	6	4	2	4
Derived	3	5	7	0	0

2.2 The analysis method

Software design has certain characteristics that are different from other engineering design disciplines. Firstly, designers often have to explore new application and technology domains that they don't have previous experience with. Therefore the quality of their design outcomes may not be consistent even for a designer who has practised for years. Secondly, a design is an abstract model and often cannot be easily judged, objectively, if it would actually work until it is implemented. So it is difficult to measure the quality of a design, especially to compare if one design is superior to another design. Because of these reasons, we do not measure design quality in this study. We note that the draft designs created by the designers make sense and the designers are experienced, so we judge that their designs would have the basic quality. Our goal is to understand the relationship between software design thinking and its influence on design effectiveness.

Design planning, problem-solution co-evolution and reasoning techniques are factors that influence design decision making. We postulate that software designers use them implicitly and in different ways during a design session. Furthermore, we suggest that designers use reasoning techniques to support design planning and decision making (see Section 4.3 for details). Figure 2a describes an iterative design decision making process model that comprises two stages. Stage 1 is when designers plan a design session to scope and prioritise the problems. Designers typically commence their design by summarizing the requirements, contextualising and scoping the problems and prioritising the issues. This kind of problem planning is breadth-first reasoning at a high-level of abstraction (Schön, 1988).

Stage 2 is when designers resolve design problems in a co-evolving manner. This is typically an iterative process of identifying design problems and solving design problems, i.e. problem-solution co-evolution. During problem-solution co-evolution, designers may carry out further design re-planning (see Figure 2a). Figure 2b lists the activities of design planning and problem-solution co-evolution. All of these activities involve decision making. Design planning involves identifying the scope of the requirements, identifying key design
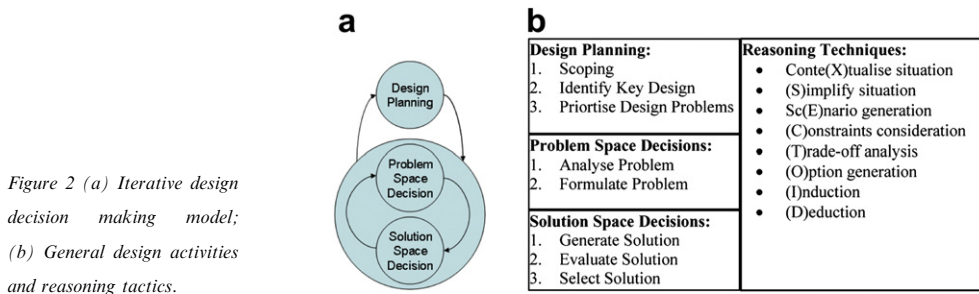

Figure 2 (a) Iterative design decision making model; (b) General design activities and reasoning tactics.

issues and their relative priorities. The problem space decision making involves problem analysis and formulation. Solution space decision making is about generating solutions and other potential options, evaluating between different options and selecting the most appropriate solution.

Figure 2b also lists the general reasoning techniques that a designer uses to support these decision making activities (Tang & Lago, 2010). A reasoning technique can be used by multiple design decision steps. For instance, a designer may identify the contexts (i.e. contextualising) for design scoping as well as for problem analysis. In making a decision, one or more reasoning techniques can be used in conjunction, i.e. they are orthogonal. A designer may use a scenario to analyse and illustrate how a solution may work, and at the same time use inductive reasoning to generate a new design idea. Also, a designer may simultaneously consider the context of a situation and the constraints that exist in order to formulate a design problem. Many examples of applying multiple reasoning techniques together are evident in this study.

In this study, we encode the protocol using the activities list and the reasoning techniques shown in Figure 2b. By analysing the protocols, we aim to understand different design thinking and how they influence design effectiveness.

2.2.1 Protocol analysis and coding schemas

We use two protocol coding schemes to analyse the designers' dialogue, using a process similar to (Gero & Mcneill, 1998) and (Yin, 2003). This coding scheme allows us to analyse the reasoning and decision making processes by the designers. Table 2 is an example of the first reasoning protocol encoding scheme. We codify the transcripts into problem space (PS) and solution space (SS) (see 3rd column in Table 2). This enables us to study the co-evolution of problem and solution. Column 4, 5 and 6 show the decision activities that the designers were taking. They correspond to the design planning and design problem-solution co-evolution process activities described in Figure 2a. The encoding classifies each statement made by a designer to indicate the type of process activity that took place. Column 7 shows the reasoning techniques

Table 2 An example of the reasoning coding scheme.

Dialogue Time	Conversation	Design Space	Planning Decisions	Problem Decision	Solution Decision	Reasoning Technique	Explicit/ Implicit Reasoning
[0:09:24.7]	We have some constraints on the flow of the queues because we can't have lights coming in a collision course	PS		Analyse Problem		Conte(X) tualise; (C)onstraint; (D)educe	(E)xplicit

that have been used. Column 8 shows whether the designers implicitly or explicitly perform reasoning. Explicit reasoning took place when the designers explicitly explained their decisions.

In parallel, we also created a second protocol to study how each design team makes decisions with respect to different decision topics. This protocol represents a set of design rationale for each design using the rationale-import mechanism in the SEURAT (Software Engineering Using RATionale System) (Burge & Kiper, 2008). The SEURAT analysis classifies the decisions and the design alternatives into a hierarchical structure based on decision topic. A sample of this hierarchy is shown in Table 3. All decision topics are time stamped for analysis. This coding scheme enables us to analyse decision making based on decision topics and time; it also allows us to analyse if decisions have been reached to address the requirements.

2.2.2 Decision map

After encoding with the first protocol, we built a decision map to represent the design activities. The decision map is based on the AREL model where design decision making is represented by three types of nodes, i.e. *design concern*, *design decision*, *design outcome*, and the causal relationships between them. A *design concern* node represents information that is an input to a design problem, such as requirements, context or design outcome. A decision node represents the design problem. A design outcome node represents the solution of a decision. A decision map shows the progression of a design by connecting the problem space (decision node in rectangular boxes without double vertical lines) to the solution space (design outcome in ovals). The contexts of the design are represented by rectangular boxes with double vertical lines. The nodes in the decision map are linked together by (a) the sequence of discussions, i.e. time; (b) the subject of the discussion. Figure 3 shows Team M's design planning in the first 5 min of their discussions. The decision map depicts the following: (a) structuring of a design approach; (b) sequencing of design reasoning; (c) design problem and solution.

3 Analysing the design activities

Using the two coded protocols and the decision maps described in the previous section, we analyse how teams A and M make decisions and how they reason with their designs. The analysis is based on three perspectives described earlier:

Table 3 An example of the coding scheme.

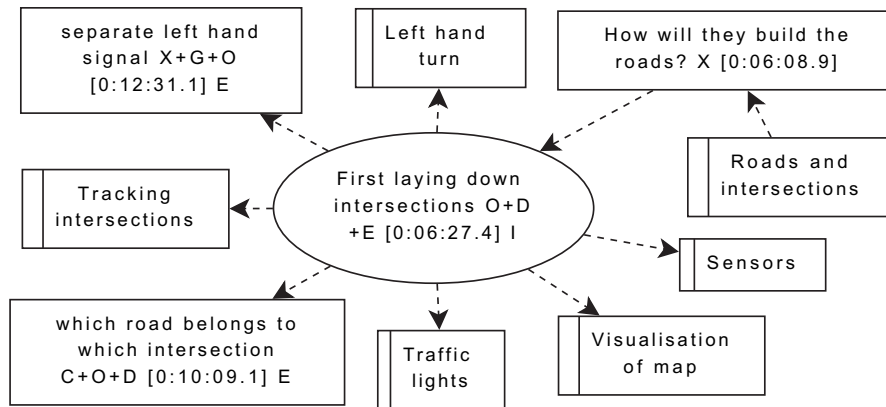| Decision ID	Time	Level	Desc
1	5	1	| | (Dec) Category: Car
1,1	5	2	| | | (Dec) IMP: See car status
1.1. (Alt)	5	3	| | | | (Alt) And it might be that we want to select a car and watch that particular car snake through the simulation (Adopted)

Figure 3 An extract of a decision map.

(a) Design planning — the question is how designers plan their design sessions and its impact on design effectiveness; (b) Design problem-solution co-evolution — the question is how designers create their design solutions and if they follow a pattern of problem-solution co-evolution, and how that impacts design effectiveness; (c) Application of design reasoning techniques — the question is what reasoning techniques are used in the above two activities and how that impacts decision making and its effectiveness.

3.1 Design planning

We suggest that the decisions on how to plan a design session have bearings on the results and its effectiveness. The considerations are the scope of a design and the priorities that should be given to each design problem. The decisions on these matters dictate the design content and therefore influence the effectiveness of design.

3.1.1 Initial design planning

Both teams started off by planning their design approach, which involves scoping the requirements, identifying and structuring high-level design problems and prioritising the problems. Team A spent about 1.5 min ([0:05:11.0] until [0:06:41.1]) on planning and they identified representation and modeling of traffic flow, signals, and intersections as their key design issues. Team M spent about 9 min ([0:06:08.9] until [[0:15:11.7]) on planning and they identified thirteen design issues. The two teams spent 1% and 9% of their total time, respectively, on problem structuring. Furthermore, there were 16 instances of scoping and 18 instances of identifying key design issues during Team M's planning, as compared to 5 and 9 instances in Team A's planning.

The difference in the way they approach design planning has bearing on how each team structures the rest of their design. Up-front identification of specific areas where design is needed dictates the structure of the rest of design discussions, even though neither team explicitly said that they were planning. For

instance, Team A started off by asserting that the solution is Model View Controller (MVC) pattern. Their focal points were data structure, modeling and representation. They spent the next 40 min ([0:08:28.4] until [0:48:02.1]) on queue management before moving to the other discussion topics. Design re-planning was carried out by Team A during the design. At [1:17:59.15], Team A assessed their design status and then realigned their focus on the design outcomes, and the rest of the design discussions followed this direction. Team M spent considerable more time exploring their plan. As a result, they first designed the layout of the road [0:16:09.7] and then they considered the other design issues subsequently.

3.1.2 Structuring design activities

Using the discussion grouping by design topics, we observe that the way each team explored the design space followed different reasoning structures. Team A used a *semi-structured approach*. They started with a design discussion that focused on the premise of a queue and the push/pop operations, and they explored this solution concept extensively to design the traffic simulation system. Further, they continued the discussions about roads and cars ([0:42:40.4] until [0:52:15.7]) but without reaching either a solution or a clear identification of the next design problem, they moved to the topic on sensors ([0:54:33.6]). Such inconclusive design discussions continued until [1:17:59.15] when they stopped and re-planned their design approach. At that point the designers re-assessed their design problems that guided them till the end of the design session. From that point, three topics, namely road modeling, simulation display and code structure were discussed.

Team M used a *major-minor structuring approach*. Team M tackled the major design problems first and when they concluded the design of those major problems, they addressed the minor associated issues afterwards. The major and minor issues are concepts in terms of the difficulties of resolving, not in terms of their importance to the system or end-users. For instance, Team M started their design discussions by addressing the issue of traffic density [0:13:35.5]. At the end of tackling the high-level design, they investigated the relevant but simpler issue regarding the simulation interface [0:26:43.5]. Then they moved to another high-level critical problem on traffic lights, and at the end of the discussions they again considered the simpler interface issue [0:41:13.3]. In doing so, the designers started with a major design problem, solved it and then tackled other minor but related design problems. In summary, the initial structuring and planning provides guidance to how design proceeds in the remaining time.

3.1.3 Design decision topic context switching

Designers in Team A and M had to deal with many design decision topics and issues that are inter-related. The focusing or switching of designers' discussions on these topics influence the way they explore the problem and solution spaces.

We call this *context switching*. Context switching takes place under two circumstances. Firstly, the designers think of related issues during a discussion and switch topic to pursue a new issue instead of the first one. Secondly, the designers may have solved an issue and then switch to a new topic.

Context switching is most noticeable in the semi-structured approach employed by Team A. For instance, during the queue management discussion, the designers focused and defocused on traffic lights several times at [0:20:36.2], [0:39:17.8] and [0:49:41.0]. Then this topic was raised again at [0:55:28.8] and [1:49:44.0] but the Team A designers did not have a clear and complete design for all issues related to traffic lights. It appears that the context switching affects the resolution of design issues. The more context switching takes place, the less the designers follow a single line of thought to explore a design issue thoroughly because the related context of the previous discussions may be lost. Therefore, even when the same issue was discussed many times, a clear solution was not devised. In contrast, Team M addressed traffic lights [0:28:16.4] in a single discussion. There is limited context switching during the discussions of these closely related design issues, e.g. sensors [0:28:54.1]. The designers acknowledged the other design issues but quickly returned to the main topic.

We observe context switching in decision making by using the decision topic protocol generated from SEURAT. In this protocol encoding, all statements made by designers are grouped by the decision topics, for instance, cars and road networks. Each topic has sub-topics organised hierarchically in multiple levels, for instance the sub-topics of *car* are *car status*, *wait time per car* etc. Context switching happens when the discussion changes from one decision topic to another decision topic at the top level. We plot the first two levels for Team A and Team M, each symbol in Figure 4a and b represents some discussions of a topic or a sub-topic. A sub-topic appears in a slightly elevated position than its corresponding topic. The *x*-axis is the timeline when the decision topics were discussed.

We have analysed the average number of context switches. Team A and Team M made 312 and 518 total statements, respectively. Out of the statements that they had made, we have found that Team A and Team M had switched discussion topics 78 times and 69 times, respectively. Represented as percentages of switching topics using the total design statements as a basis, Team A has a 24.9% chance of switching to a new topic and Team M has a 13.2% chance of switching to a new topic. Therefore, Team M is more focused on their discussion topics and perform less context switching compared to Team A.

3.1.4 Time distribution of design discussions
It is observed in Figure 4a that Team A discussed 9 different decision topics and some of the discussions spanned across the entire design sessions,
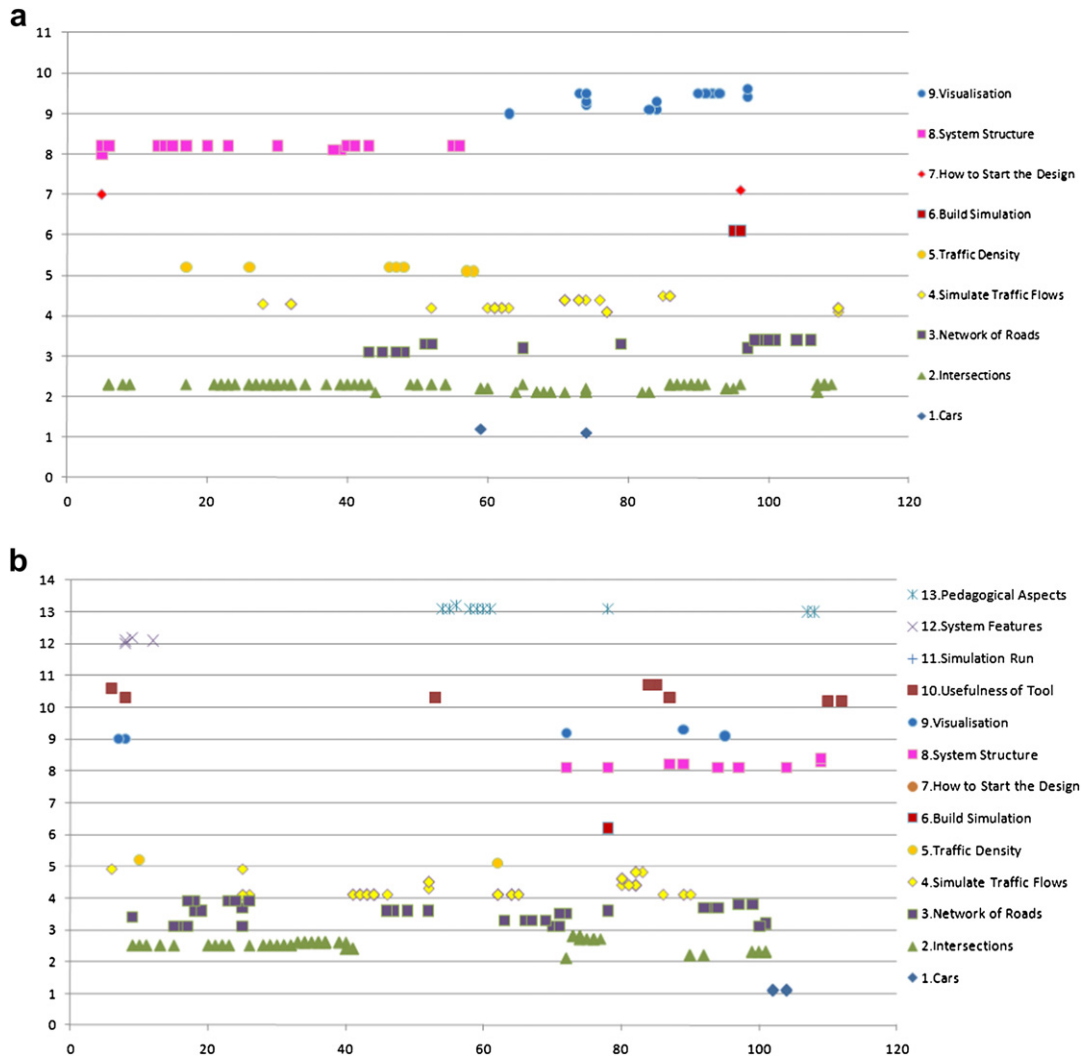
Figure 4 Decision Topics Time Distribution: (a) Team A (b) Team M.

especially topics 2, 4 and 8. Team M discussed 12 topics. The time distribution of discussions on topics 2, 3 and 4, are quite different between Team A and M. Team M's discussions in each topic are clustered together in a timeline (Figure 4b). To enable us to quantify the distribution of time on the discussions of each decision topic, we have computed the standard time deviation (in minutes) for each decision topic (see Table 4). As shown in Figure 4b, discussion topic 2 has three clear clusters. This means that the designers focused on this topic in three different periods. In this case, the standard deviation is computed for each cluster to reflect the concentration of the discussion. If there is no obvious clustering then all the data points are used in computing the standard deviation.

What makes software design effective? 625

Table 4 − Standard Deviation in Time by Decision Topics.

	1	2	3	4	5	6	7	8	9	10	12	13
Team A		13.9	2.8	21.9	5.6	0.5		12.5	7.9			
Team M	1.0	5.2	3.3	1.9				10.6	2.5	1.2	2	1.4

The blank cells in Table 4 indicate either the team did not discuss this topic or we forego the data because there are too few data points to compute the distribution meaningfully. For instance, topic 7 was discussed by Team A only twice, the two data points give a standard deviation of 64 min. The results show that Team A has spent considerably more elapsed time discussing topics 2, 4, 8 and 9. When there are many topics that have a high time distribution, it indicates that the topic discussions overlap each other in the same timeline. The overlapping of discussion increases the switching between topics.

One may consider that a decision topic is particularly difficult and therefore requires a longer time to solve. However, if we compare the standard distribution of discussion time between the two teams on all topics, Team M consistently has smaller time distributions than Team A, indicating that Team M finished dealing with decision topics more quickly. If we examine the requirements coverage (see Section 5.2) together with the time distribution, we see that Team M has addressed more requirements than Team A in total and each topic has a shorter time span. This data suggests that the frequent switching of discussion topics by Team A has made the discussion less effective. Team A spent more time on a topic because the discussion was switched back and forth before the issues were fully resolved. When this happens, it is possible that the context and the issues of a previous discussion have to be reestablished. An example is that a designer in Team A said that 'I'm sort of reminded of the fact that we're supposed to allow lights to have sets of rules…' at [0:49:41.0]. This was the third time they discussed the same design topic.

3.2 *Problem-solution co-evolution*

Dorst and Cross have observed co-evolution of design between problem and solution spaces (Dorst & Cross, 2001). The way the designers in the two teams conducted their designs was largely aligned with the co-evolution model. For instance, Team M demonstrated a co-evolution process where a problem was identified and solved, leading to the next problem-solution cycle (Figure 5). However, this model presumes a logical design reasoning behavior in which a solution is derived from well-defined design problems. We have observed design behaviors in this study that differ from this model.

At a very early stage of design, Team A identified traffic flow and signals as design issues [0:06:10.6] and they decided that a queue should be used in the
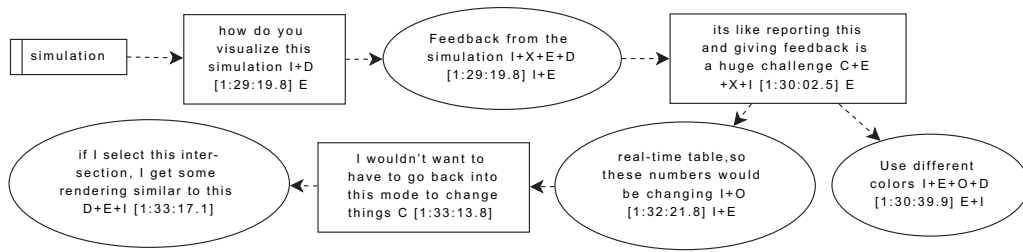
simulation

how do you visualize this simulation I+D [1:29:19.8] E

Feedback from the simulation I+X+E+D [1:29:19.8] I+E

its like reporting this and giving feedback is a huge challenge C+E +X+I [1:30:02.5] E

Use different colors I+E+O+D [1:30:39.9] E+I

real-time table,so these numbers would be changing I+O [1:32:21.8] I+E

I wouldn't want to have to go back into this mode to change things C [1:33:13.8]

if I select this inter-section, I get some rendering similar to this D+E+I [1:33:17.1]

Figure 5 Example of a problem-solution co-evolution in Team M design session.

solution [0:08:28.4]. From this point onwards, Team A continued to make a series of decisions surrounding the operations of the queue. Some examples are [0:08:51.0], [0:38:05.6] and [0:39:17.8]. The idea of the queue drove the discussions from [0:08:28.4] until [0:50:43.1]. The design plan and all subsequent reasoning are predominantly based on this solution concept that was decided up-front, without questioning whether this is the right approach or if there are alternative approaches. It is also evident that Team A did not spend much time exploring the problem space and they may have made implicit assumptions about what the solution would provide. This case demonstrates that when designers take a solution-driven approach, the preconceived solution becomes the dominant driver of the design process, and the rest of the solutions evolves from the preconceived solution. Such design behavior hints at the dominance of cognitive System 1, where designers autonomously selected the MVC and queue as the solution, over System 2, the reasoning system.

There are possible reasons for this situation. Firstly, it has been observed that when people make estimates to find an answer from some initial values, they tend not to adjust to another starting point but instead are biased towards the initial values. This is called the anchoring and adjustment heuristic (Epley & Gilovich, 2006; Tversky & Kahneman, 1974). Similarly, Rowe (Rowe, 1987) observed that a dominant influence was exerted by the initial design even when severe problems were encountered. Designers are reluctant to change their minds. Tang et al. also found that for some software designers, the first solution that came to mind dominated their entire design thinking, especially when the software designers did not explicitly reason with their design decisions (Tang et al., 2008). Secondly, a designer may be forced to make premature commitments to guess-ahead a design, but the design commitment remains. (Green & Petre, 1996).

3.2.1 Problem space decisions

Deciding on what design problems need solving can be made implicitly or explicitly. Team A generally took a solution-driven approach defining their problem space implicitly. Team M, on the other hand, explicitly stated their

problem statements, they typically analysed the relevant information to the problem such as goals, resources and current situations of a system. Designers who work explicitly in the problem space typically determine what information is known and what is unknown, and based on that decide what to design for (Goel & Pirolli, 1992). We encode explicit problem statements, distinguishing between problem analysis and problem formulation. Problem analysis is a step to identify the information relevant to a problem. It provides the context of a problem. Problem formulation is a step to articulate what problems need to be solved or how to proceed to solve them.

Team A and Team M made 49 and 46 analysis statements, respectively. However, Team A only formulated 34 problem statements whereas Team M formulated 120 problem statements. That represents 13.4% and 31.9% of all the statements made by the two teams, respectively. In a study of general problem solving, (Goel & Pirolli, 1992) found that problem structuring accounts for between 18% and 30% of all design statements made in a problem solving exercise. Furthermore, Team A and M made 153 and 173 solution statements, respectively. We plot the problem formulation and solution generation across
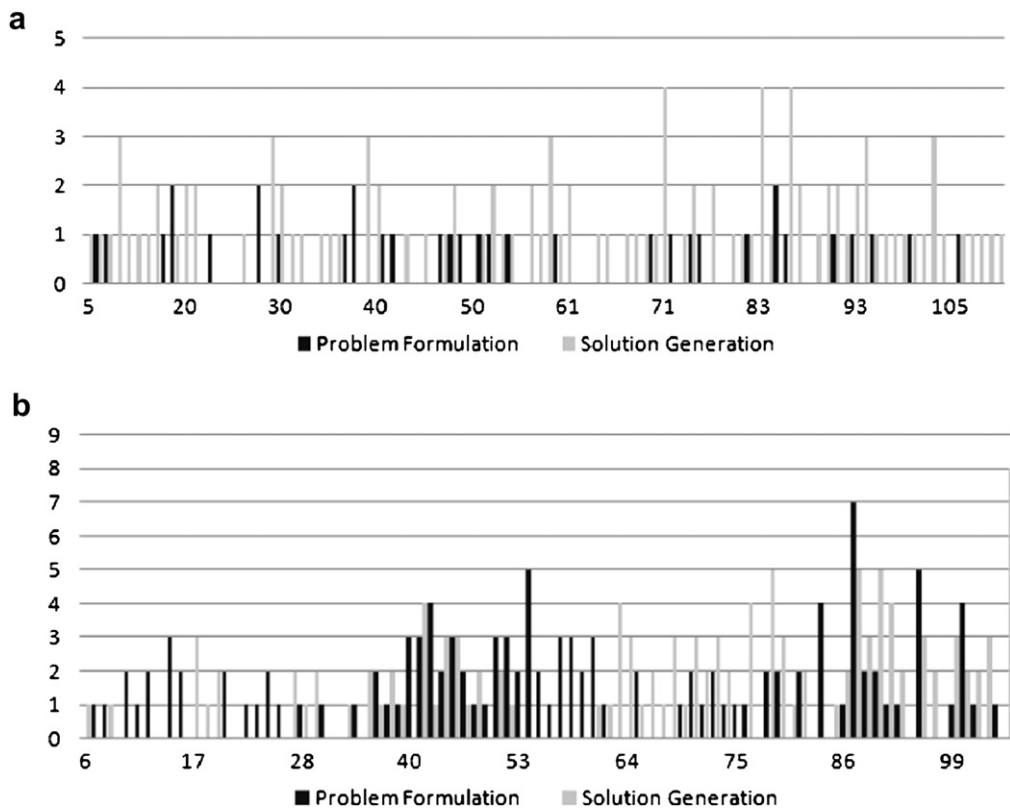

Figure 6 Problem Formulation and Solution Generation in a Design Session: (a) Adobe (b) AmberPoint.

time in a graph for both design session (Figure 6). The Problem Formulation columns indicate the frequency of problem formulation statements in 1 min; the Solution Generation columns indicate the frequency of solution generation statements in 1 min. The concentration of problem analysis and formulation by Team M is high (Figure 6b), especially in certain periods, e.g. between [0:12] to [0:16] and [0:30] to [1:02]. On the contrary, Team A has fewer problem statements (Figure 6a).

3.2.2 Solution space decisions

To analyse decision making in the solution space, we divide the decision process into three general steps: generating a solution option, evaluating the pros and cons of the available options when there is more than one solution option, and selecting a solution from the available options. In encoding the protocols, Team A and M made 106 and 124 solution generation statements, 17 and 16 evaluation statements, 30 and 32 selection statements, respectively. The number of solution statements made by the two teams is very similar. However, the process in which the two teams took is quite different. Problem and solution statements are encoded in the protocol analysis (see Figure 2b).

Analysing the problem and solution statements that the two teams made shows that the ratios of all solution statements to all the statement made by Team A and M are 60.2% and 45.7%, respectively. It means that for Team A, 6 out of 10 statements are solution statements. In total, Team A made 153 solution statements out of a total of 254 statements, whereas Team M made 172 solution statements out of a total of 376 statements. Given the 2 h of design time, Team A made a similar number of solution statements to Team M but their solutions statements represent a higher proportion of their design dialogue, implying that they are spending more time on devising the solutions.

Additionally, the ratios of solution statement generation to problem statement formulation made by Team A and M are 32% and 96.7%, respectively. Team A has a lower ratio of design problem formulation to solution generation, indicating that they generate solutions with less explicitly identified problems. Although there is no benchmark on the sufficiency of problem space formulation, it is suggested that a clearly defined problem space is important (Maher et al., 1996). In the case of Team A, they have a lower ratio of formulated problem and a higher ratio of generated solution. We typify this kind of design strategy as a solution-driven strategy. On the other hand, Team M has a ratio which is close to one, indicating that, on average, for every problem they posed, they also generated a solution.

3.2.3 Interleaving problem-solution decisions

The Mahler et al. (Maher et al., 1996) and Dorst and Cross (Dorst & Cross, 2001) co-evolution models suggest that design evolution takes place along

two parallel lines: problem space and solution space. Tang et al. suggest that software design evolves linearly between problem and solution in a causal relationship (Tang et al., 2009). These models follow a fundamental premise that solutions are generated from problems. However, the evidence from the Team A design session has suggested that some software designers employ a solution-driven decision making approach in which problem formulation plays a lesser role (see Figure 6a). Designers can devise solutions even though the problems are not well articulated. This may be due to the nature of software design in which the goals and sub-goals of a design can be abstract. Therefore designers can arbitrarily decide how much of the problem space to explore before delving into the solution space.

The co-evolution models also suggest that the development of the problem space and the solution space progress in parallel. However, the behavior of the designers depicted in Figure 6b suggests that other activities are also involved. Firstly, there are extended periods when Team M formulated a lot of questions, at [0:12] and [0:53]. This indicates that the designers explored the problem space intensively for some time without producing any solutions. The designers analysed and formulated inter-related problems to gain a comprehensive perspective of the domain. The decision topics protocol analysis (Figure 4b) shows that Team M was focusing on intersection arrangement from [0:12] and traffic flow simulation from [0:53]. Secondly, after intense explorations of the problem space, the designers follow the general co-evolution model in which the problem decisions and solution decisions statements are interleaved, indicating that the designers had decided on what problems to solve and then proceeded to solving them.

The activities carried out by Team M demonstrate that problem-solution co-evolution would happen at different levels. When the designers are at the exploratory stage, all related problems are explored to gain a better understanding of the problem space. At this stage, few solutions are provided. When all the design issues are well understood, designers would then hone in on a single design issue and proceed to creating a design with problems and solutions co-evolution. This approach seems to serve Team M quite well in terms of exploring the problem space effectively.

3.3 Design reasoning techniques

Design is a *rational problem solving* activity—it is a process in which designers explore the design issues and search for rational solutions. It is well established by many that design rationale is one of the most important aspects in software design (Shum & Hammond, 1994; Tang et al., 2006). However, the basic techniques that lead to rational design decisions are not well understood. Therefore, it is necessary to explore different design reasoning techniques that are used by the design teams. The initial list of reasoning techniques (Figure 2b) is derived from the design rationale literature (Tang et al., 2006; Tang &

Lago, 2010; Tyree & Akerman, 2005) and various literature that describe how software design is to be conducted. First of all, there is a general creative process and some reasoning process that goes on. We encode the protocol with inductive and deductive reasoning. Inductive reasoning, especially analogical reasoning, is a means of applying knowledge acquired in one context to new situations (Csapó, 1997). It promotes a creative process in which issues from how different pieces of information can be combined to form a new artifact (Simina & Kolodner, 1997). There are different techniques of inductive reasoning (Klauer, 1996) that software designers can use.

For instance, one technique is scenario-based analysis (Kazman, Abowd, Bass, & Clements, 1996). It describes a process of using scenarios to evaluate system architecture. Another technique is contextualisation, where identifying the context at the right level of abstraction helps to frame the design issues (Simon & Newell, 1972). We conjecture that reasoning techniques are fundamental to the design decision making process, so they can be used in design planning, problem and solution decision making. In this section, we discuss the reasoning techniques and how they are used by the two teams.

3.3.1 Contextualising problems and solutions

Contextualisation is the process of attaching a meaning and interpreting a phenomenon to a design problem. Well reasoned arguments state the context of the problem, i.e. what and how the context influences the design. All teams contextualise their problems in different ways; we counted the number of statements where they discuss the context of the problem and state the details that can influence the potential solution. We identified 62 such statements, problems and solutions, by Team A and 97 statements by Team M. It means that Team M would identify the information that surrounds a problem or a solution as part of their analysis before they proceed to decide on what to do with the information. Team M is more thorough in articulating the context. An example is when Team M examines the traffic lights [0:28:16.4]. The designers spell out 3 relevant contexts of the problem, i.e. the timing of the cycle, the impact of the sensors and the left-turn arrow. Their discussion of the design problem using those contexts happened at the same time frame and as such it provides a focal point for investigating a design problem comprehensively.

On the other hand Team A contextualises their design problems in an uncoordinated way. For example, when Team A discussed traffic signals, they did not consider all relevant contexts at the same time. The designers discussed signal rules at [0:12:31.1] and they revisited the subject of traffic signals with respect to lanes at [0:25:37.4]. Later they decided that it is the intersection which is relevant to traffic signals [0:30:41.9]. At [0:56:06.7] they discussed how signals could be specified by the students. Twenty-one minutes later [1:17:58.4] their discussion went to the subject of sensor and left signal. Contextualisation influences how a design problem can be framed; the difference between how

the two teams use contextualisation affects what information is used for reasoning with a design problem. The lack of thorough contextualisation in the problem space may be one reason why Team A context switches more frequently. If the subject matter is not investigated thoroughly and design gaps are identified later, designers would revisit the same design topic.

3.3.2 Scenario construction

The use of scenarios enhances the comprehension of a problem and facilitates communication between designers. Constructing a scenario is a way to instantiate an abstract design problem and evaluate if the abstract design is fit for the real-world problem (Kazman et al., 1996). Carroll suggests that scenarios help designers to understand needs and concerns, and then make generalisation of that knowledge into a solution (Carroll, 1997). For example Team A built a scenario of how the simulation could work, talking about the possibility of having a slider to control the timing of lights and car speed [0:48:20.4]. Using this scenario, the designers highlighted a number of related design problems, e.g. the average wait time of a car going through the city streets; car speeds; traffic light control etc. Team A and Team M made 93 and 80 scenario statements, respectively. This indicates that the scenario is an important reasoning technique that both team use extensively. In this study, we do not find any significant relationship between how the designers use scenarios as a reasoning technique and the effectiveness of design.

3.3.3 Creating design options

Creating design solution options is an important reasoning technique used by expert designers (Cross, 2004). It has been found that by simply prompting designers to consider other design alternatives, designers with less experience create better designs (Tang et al., 2008). Team A stated 37 options for 8 problems, a 1:4.5 problem to option ratio. Team M stated 58 options for 20 problems, a 1:3 problem to option ratio. In total, Team M generated 36% more options than Team A, probably because of the higher number of design problems identified.

We have observed certain characteristics about generating design options: (a) Problem refinement − designers contemplated the problem and the solution together to gain a better understanding of the design problem. Designers would use phrases such as '*But you would have to deal with this issue and you may also do this*'. Designers would generate solution options to validate the design problem; (b) Constraint specification − when a constraint(s) is considered, a new solution option is sometimes generated to overcome the constraint. Designers would use phrases such as '*but it depends on this*' or '*it cannot do this*'; (c) Scenario driven − a new design option can be generated when designers consider different scenarios. Designers would use a phrase such as '*Maybe we have … a scenario … you would want to do this*'. Although these can be observed during design, they were used subconsciously by

designers. Designers did not explicitly ask '*What other design options may we have*?'. Such observation suggests that the designers did not consciously consider option generation as a helpful technique in finding an appropriate design solution.

3.3.4 Inductive and deductive reasoning

Inductive reasoning is a type of reasoning that generalises specific facts or observations to create a theory to be applied to another situation whereas deductive reasoning uses commonly known situations and facts to derive a logical conclusion. While inductive reasoning is exploratory and helps develop new theories, deductive reasoning is used to find a conclusion from the known facts.

Let us consider first how both teams have used inductive reasoning. We have counted 71 and 112 induction statements from Team A and Team M, respectively. Team M has used 36% as many inductive reasoning statements as Team A. It suggests that Team M employs a more investigative strategy, inspecting the relevant facts to explore new design problems and solutions. As an example, Team M discussed the relationships between the settings of traffic lights to prevent collision [0:34:30.6]. From the observations that cars can run a yellow light and to prevent collision, they induce that there should be a slight overlap when all traffic lights facing different directions should be red simultaneously. Then they reason inductively about the manual settings for the timing of the lights and whether the users should be allowed to set all four lights manually. Since the users should not be allowed to set the traffic lights to cause collisions, the designers reason that there should be a rule in which the setting of a set of lights would imply the behavior of the other traffic lights. Team M in this case found new design problems from some initial observations.

On the other hand, deductive reasoning is used more times by Team A than Team M, with 56 and 51 statements respectively. For instance, Team A started off by predicating on the basic components used in the solution, i.e. queue and cop etc., and deduced the solution from that. They used deductive reasoning at [0:30:19.3]. They said that a lane is a queue, so multiple lanes would be represented by multiple queues. Since they also predicated on the use of a traffic cop, so they deduced that the cop would look after the popping and pushing of multiple queues. Team A was fixed in the basic design of the system from the beginning without exploring other design options.

3.3.5 Explicit reasoning

Explicit reasoning is the use of explicated arguments to support or reject a design problem and a solution choice. Implicit design reasons are decisions made without communicating an explicit reason. When designers use explicit reasoning, they often use the words *because* or *the reason*. We suggest that explicit reasoning helps designers to communicate their ideas better. When explicit reasons are absent, people other than the designers themselves make

assumptions about why the decision takes place. In this study, explicit reasons can be identified with the word 'because' in the protocol and the arguments to support the explanations. Team M made 33 cases of implicit reasoning and 87 cases of explicit reasoning, Team A made 47 cases of implicit reasoning and 52 cases of explicit reasoning.

4 Design decision making and design effectiveness

Software design experience is very important but it is also domain specific. In this experiment, designers are highly experienced in software design and they possess general knowledge about the problem domain, but they do not appear to have detailed design knowledge in this area. So the designers have relied on their general design knowledge and reasoning to create design solutions. All designers were given 2 h to design, which was insufficient time to exhaustively explore all possible solution branches to provide a complete solution. Therefore, their design is measured by how effective the designers have been analysing the problems and formulating the solutions. We do not verify or validate each team's design solution or evaluate their quality because: (a) the criteria we use in design quality reviews may bias the research findings; (b) the solutions are incomplete; and (c) designers in some cases have not sufficiently explained a design decision. In order to understand how design reasoning and decision making influence their design outcomes, we investigate time utilisation and requirement coverage.

4.1 Design time effectiveness

The ways Team A and Team M spent time on organising their designs are different. This can be summarised in two ways. Firstly, Team M spent more time on planning than Team A did. This seems to have influenced how Team M's design discussions were conducted, which have more structure than Team A's. Secondly, Team M had fewer context switches than Team A. Team M's discussion topics were clustered together, focusing on the same line of thoughts and solving most problems in the same design topic before moving onto another topic. In computer processing, context switching between tasks involves computation overheads for preparing the next task. We suggest that cognitive context switching in design also involves certain overhead such as realigning the context of the new discussion topic. The efficiency and focus of design discussions is lower if extensive context switching is involved.

4.2 Requirement coverage

Requirement coverage is defined as the number of definitive solutions that are provided to address the requirements, derived requirements and design outcomes. Tables 5 and 6 show the requirements and the derived requirements coverage. Requirement coverage is the results of the analysis of the two protocols. We classified the design decisions by their status: mentioned (M), discussed (D), proposed (P), Implied Resolution (I) and Resolved(R). I and R indicate that a solution has been reached. The other statuses indicate that the topics have been visited but no decisions have been made (see Table 5).

Table 5 Requirements coverage based on decision status.

Requirement Category	Team A					Team M				
	M	D	P	I	R	M	D	P	I	R
Network of Roads		1,0		1,0	2,2				1,1	3,2
Intersections/Lights			1,0		5,5		0,1	1,0		5,4
Simulate Traffic Flows	1,0				2,7					4,7
Density					2					2
NFR					1					4

The number of requirements and derived requirements are shown in a format of [x,y] where x indicates the number of requirements coverage and y indicates the number of derived requirements coverage. Table 5 shows the number of requirements and derived requirements coverage each of the two teams has achieved. For instance, Team M has resolved 3 Network of Roads requirements and 2 of its derived requirements (see Table 5 − row 1 right hand cell). After adding all the requirements and the derived requirements that each team have accomplished (i.e. implied and resolved), we have found that Team A has achieved 13 requirements and 14 derived requirements, whereas Team M has achieved 19 requirements and 14 derived requirements (Total column in Table 6). It indicates that given the same time, Team M managed to cover more requirements and derived requirements than Team A, and they are more effective in their design approach.

4.3 Differentiating the design approaches

Based on the previous analysis, we differentiate how the two teams approach their designs. We compare and contrast them in Table 7 to characterise the different approaches. We suggest that the characteristics listed in Table 7 collectively contribute to design effectiveness. However, we cannot delineate the relative importance of individual factors. All these factors contribute to software design effectiveness in one way or another. Effective time utilisation depends on good design planning, minimal context switching and well-organised problem exploration. Additionally, it seems that a problem driven design approach supported by effective use of reasoning techniques is also essential in software design. In evaluating time effectiveness and requirement coverage, we note that both designs are potentially viable solutions. They both make sense from a software design point of view. On this basis, we study the effectiveness and the reasoning of the designers.

Table 6 Requirements coverage of Team A and M.

	Network of Roads [4,3]	Intersection/Light [6,5]	Simulate Traffic Flows [4,7]	Traffic Density [2,0]	NFR [4,0]	Total [20,15]
Team A	3,2	5,5	2,7	2,0	1,0	13,14
Team M	4,3	5,4	4,7	2,0	4,0	19,14

Table 7 Decision making and reasoning approaches of Team A and M.

	Team A	Team M
Design Planning		
Design planning at the start of the design session	Brief	Extensive
Structure of the design discussions	Semi-structured − the discussions did not follow any plan	Major-minor structure − the discussions followed the initial scope and addressed the difficult issues followed by the easier issues
Decision topics context switching	More frequent − moving from one discussion topic to the next before a decision topic was fully explored	Less Frequent − exploration of a topic was thorough
Decision topics time distribution	Widespread over time − some decision topics were revisited many times and distributed throughout the design session	Limited time distribution − decision topics discussions were focused and were not spread out
Problem-Solution Co-evolution		
Problem statement formulation	Low − the design problems are often not stated. The problems statements are sporadic.	High − problem statements were stated, and they interleaved with solution statements. There were two periods when problem statements were predominantly made to explore the situations.
Problem-solution approach	Solution driven − the concept of using MVC solution and the queue dominated the design thinking	Problem driven − explored the design questions before providing a solution
Reasoning Techniques		
Contextualise the problems and the solutions	Medium − less exploration of what factors might have influenced a design decision	High − explored the contexts of the problems and the solutions more thoroughly
Use of Scenarios	High − use scenarios as an example to understand the needs and concerns.	High
Creating the design options	Less frequent in discussing design options. Higher problem to option ratio.	More frequent in discussing design options. Lower problem to option ratio.
Inductive Reasoning	Medium − the level of generalisation as evident in the requirement coverage.	High − generalises specific facts or options from a set of observations. Designers generate new ideas from using various reasoning techniques.
Explicit Reasoning	Low − often the designers do not explain the reasons why certain solutions are chosen	High − the reasons of a choice are explicitly stated, and this mutual understanding between the designers create opportunities to explore the problem space and the solution space further

Spending design time effectively can help designers focus on design issues and analysis. Additionally, good decision making approaches and design reasoning techniques can also contribute to software design effectiveness. A summary comparison is shown in Table 8.

5 Research validity

One of the major threats to validity is the potential subjectivity of the researchers in interpreting the protocols. In this work, we have used two separate

Table 8 Design Time Effectiveness and Requirement Coverage.

	Team A	Team M
Time effectiveness	More context switching; less design problem statements were made; revisited same design topics many times.	Less context switching; focused discussions and more problem statements were made, systematic exploration of problem and solution spaces.
Requirements coverage	Less requirements are covered	More requirements are covered.

protocol encodings to analyse the design activities. Although the two protocols have different perspectives, the data obtained from them are consistent, so it provides triangulation for our interpretation. The first protocol coding was refined over a period of 6 months, in no less than 10 iterations, by two researchers. This investigation is different to previous works and the protocol encoding is new; and the interpretation of the protocols, especially on concepts such as inductive and deductive reasoning, required deliberations by the researchers. Therefore we adjusted our interpretations of the reasoning techniques according to our improved understanding in each encoding iteration. The second protocol encoding was performed over a period of 2 months, in 3 iterations, by a single researcher. This encoding was not new and the researcher had experience using it on prior projects in multiple domains.

The findings from analysing the two protocols and the decision maps have highlighted different aspects of design reasoning. Our findings in areas such as problem structuring and problem-solution co-evolution are consistent with other design theories, which provide external validation to this work.

It emerged in discussions with the designers at the workshop that the design sessions represented a very different situation from those the designers normally encounter in their work, notably in being supported by far less specialized knowledge of the problem setting and context. Such lacking of domain knowledge may have influenced they way the designers acted during design and thus the resulting design.

Without assessing the quality of a design, we only note that both of the designs are potentially viable solutions. The threat to this research is we may judge a team effective when they produce a solution that does not fit the requirements. The need for making such an assumption is because given the short design session and the draft design, it is impossible to evaluate design quality fairly. Nevertheless, this assumption allows us to have a baseline for evaluating design effectiveness.

6 Conclusion

Design decision making and reasoning play a major role in software design, but there is a limited understanding on how they work, whether one way of doing things can help designers to design more effectively than another way.

Two teams of designers were given non-trivial software design problems. We studied the ways they make design decisions and reasoning from the perspectives of planning, time utilisation and reasoning. We have found that proper planning of design discussions improves design issue identification and facilitates the searches for solutions. The approach to spending time on decision topics also impacts on time utilisation. Excessive context switching of decision topics would defocus the design discussion, and thus designers require more time to address all relevant design issues. An example is that the repeated discussions on the same topic made by Team A are less effective compared to the single and focused design discussion made by Team M on the subject of traffic lights.

We have found that software designers do not always use the problem-solution co-evolution approach. Some designers autonomously assume that the first solution they come across is the right solution and they anchor on that solution all the way without justifying their initial decision. Some designers follow a problem-solution co-evolution and a rational reasoning approach. It appears that in dealing with an unfamiliar domain. A reasoning approach (System 2) is better than an autonomous approach (System 1) because of better problem space exploration and thorough consideration of solution alternatives.

We have also found that reasoning techniques, such as appropriate contextualisation of design problems, explicit communication of design reasoning, explicit design reasoning and the use of inductive reasoning contribute to the effectiveness of software design. We have found examples to demonstrate that the exploration of design problems may be related to the level of inductive reasoning, suggesting that inductive reasoning can play a role in defining the design problem space. For instance, Team A generated fewer inductive reasoning statements (Section 4.3.4), they also explored relatively fewer problems (Section 4.2) and they employed a solution-driven approach (Section 4.2.3). If inductive and deductive reasoning is complemental, and if problem and solution formulation is complemental, the use of a predominantly deductive and solution-driven approach may have a negative influence on how much of the problem space a designer explores. Although this is an interesting observation, the evidence that we have here to support it is insufficient and will require further research.

From this study, we have preliminary evidence to show that design decision making that is based on good design planning, minimal context switching, well-organised problem exploration and good reasoning can help effective software design. As this is an exploratory study, further in-depth empirical and experimental studies of these aspects can improve the way we design software. This is fundamental to the software development process.

Acknowledgements

1. This requirement coverage is from the analysis made by the authors, which can be different to the other analyses. Details can be found in a supporting document: http://www.users.muohio.edu/burgeje/SPSD/SPSD_Requirements_Coverage.pdf

References

Burge, J., & Kiper, J. (2008). Capturing collaborative design decisions and rationale. *Proceedings of design, computing, and cognition*. Springer.

Carroll, J. (1997). Scenario-based design. In M. Helander, T. K. Landauer, & P. Praphu (Eds.), *Handbook of Human-computer Interaction*. Elsevier.

Cross, N. (2004). Expertise in design: an overview. *Design Studies, 25*, 427−441.

Csapó, B. (1997). The development of inductive reasoning: crosssectional assessments in an educational context. *International Journal of Behavioral Development, 20*, 609−626.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31*, 1268−1287.

Dorst, K., & Cross, N. (2001). Creativity in the design space: co-evolution of problem-solution. *Design Studies, 22*, 425−437.

Dutoit, A., Mccall, R., Mistrik, I., & Paech, B. (Eds.). (2006). *Rationale management in software engineering*. Springer.

Epley, N., & Gilovich, T. (2006). The anchoring-and-adjustment heuristic. *Psychological Science, 17*, 311.

Evans, J. (1984). Heuristic and analytic processes in reasoning. *British Journal of Psychology, 75*, 451−468.

Evans, J. S. (2003). In two minds: dual-process accounts of reasoning. *Trends in Cognitive Sciences, 7*, 454−459.

Gero, J., & Mcneill, T. (1998). An approach to the analysis of design protocols. *Design Studies, 19*, 21−61.

Goel, V., & Pirolli, P. (1992). The structure of design problem spaces. *Cognitive Science, 16*, 395−429.

Goldschmidt, G., & Weil, M. (1998). Contents and structure in design reasoning. *Design Issues, 14*, 85−100.

Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: a 'Cognitive Dimensions' framework. *Journal of Visual Languages and Computing, 7*, 131−174.

Hall, J. G., Jackson, M., Laney, R. C., Nuseibeh, B., & Rapanotti, L. (2002). Relating software requirements and architectures using problem frames. *IEEE Joint International Conference on Requirements Engineering*.

Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, J. H., Ran, A., & America, P. (2007). A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software, 80*, 106−126.

Kazman, R., Abowd, G., Bass, L., & Clements, P. (1996). Scenario-based analysis of software architecture. *IEEE Software, 13*, 47−55.

Klauer, K. J. (1996). Teaching inductive thinking to highly able children. In A. J. Cropley, & D. Dehn (Eds.), *Fostering the growth of high ability: European perspectives*. Greenwood Publishing Group.

Maher, M. L., Poon, J., & Boulanger, S. (1996). *Formalising design exploration as co-evolution: A combined gene approach. Key centre of design computing*. Sydney: University of Sydney.

Rittel, H. W. J., & Webber, M. M. (1973). Dilemmas in a general theory of planning. *Policy Sciences, 4*, 155−169.

Rowe, P. G. (1987). *Design thinking*. Cambridge, Mass: MIT Press.

Schön, D. A. (1988). Designing: rules, types and worlds. *Design Studies, 9*, 181−190.

Shum, B. S., & Hammond, N. (1994). Argumentation-based design rationale: what use at what cost? *International Journal of Human-Computer Studies, 40*, 603−652.

Simina, M., & Kolodner, J. (1997). *Creative design: Reasoning and understanding. Case-Based Reasoning Research and Development*. Springer Berlin/Heidelberg.

Simon, H., & Newell, A. (1972). *Human problem solving: The state of the theory in 1970*. Carnegie-Mellon University.

Tang, A., Han, J., & Vasa, R. (2009). Software architecture design reasoning: a case for improved methodology support. *IEEE Software*. Mar/Apr 2009. 43−49.

Tang, A., & Lago, P. (2010). *Notes on design reasoning techniques. Swinburne FICT technical reports*. Melbourne: Swinburne University of Technology.

Tang, A., Barbar, M. A., Gorton, I., & Han, J. (2006). A survey of architecture design rationale. *Journal of Systems and Software, 79*, 1792−1804.

Tang, A., Tran, M. H., Han, J., & Van Vliet, H. (2008). Design reasoning improves software design quality. *Proceedings of the Quality of Software-Architectures (QoSA 2008)*.

Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: heuristics and biases. *Science, 185*, 1124−1131.

Tversky, A., & Kahneman, D. (1986). Rational choice and the framing of decisions. *The Journal of Business, 59*, S251−S278.

Tyree, J., & Akerman, A. (2005). Architecture decisions: demystifying architecture. *IEEE Software, 22*, 19−27.

Yin, R. (2003). *Case study research design and methods*. Sage Publications.

Zannier, C., Chiasson, M., & Maurer, F. (2007). A model of design decision making based on empirical results of interviews with software designers. *Information and Software Technology, 49*, 637−653.