VU Research Portal

In Two Minds: How Reflections Influence Software Design Thinking

Razavian, M.; Tang, A.; Capilla, R.; Lago, P.

2015

[Link to publication in VU Research Portal](#)

citation for published version (APA)

Razavian, M., Tang, A., Capilla, R., & Lago, P. (2015). *In Two Minds: How Reflections Influence Software Design Thinking*. VU University Amsterdam.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

E-mail address:

vuresearchportal.ub@vu.nl

In Two Minds: How Reflections Influence Software Design Thinking

Maryam Razavian, VU University Amsterdam
Antony Tang, Swinburne University of Technology
Rafael Capilla, Universidad Rey Juan Carlos
Patricia Lago, VU University Amsterdam

Technical Report 2015-.001

April 15, 2015

Contents

0.1	Introduction	
0.2	Background: Software Design Thinking and Reasoning	
0.2.1	Human Issues	
0.2.2	Design Reasoning Approaches	
0.2.3	Design Reflection	
0.3	Reflection and Reasoning	
0.3.1	Mind 1: Design Reasoning and Decision Making	
0.3.2	Mind 2: Design Reflection	
0.4	Methodology	
0.4.1	Research Approach	
0.4.2	Designing the Exploratory Experiment	
0.4.3	Experiment Execution	
0.4.4	Data Analysis	
0.4.4.1	Codes	
0.4.4.2	Coding Procedure	
0.4.4.3	Coding Consistency	
0.4.4.4	Qualitative Causality Analysis	
0.5	Results	
0.5.1	Finding 1 - External Reflection Induces Internal Reflection	
0.5.2	Finding 2 - Reflection Improves Quality of Design Discourse	
0.5.2.1	Finding 2.1 - External reflection improves quality	
0.5.2.2	Finding 2.2 - Without internal reflection, design quality does not propel	
0.5.3	Finding 3 - Reflection Cannot Improve Quality without Domain Knowledge	
0.6	Limitations	
0.7	Discussion	
0.8	Conclusion	
Appendix A	Diagrams	

List of Figures

1	Software Design Thinking Model - Mind 1 and Mind 2
2	Design discourse and reflections and in T1
3	Quality of design discourse in T1
4	Design discourse and reflection in T3

List of Tables

1	Reflective questions used by Mind 2 to reflect on Mind 1
2	List of codes
3	Explanatory effect matrix of Transcript 1
4	Transcript 1 Interpretation
5	Teams that had internal reflections.
6	Number of reflections by test teams and control teams
7	Transcript 2 Interpretation
8	Transcript 3
9	Transcript 4
10	Transcript 5 Interpretation
11	Transcript 6 Interpretation
A-1	Reflection and quality of design discourse in test teams
A-1	Reflection and quality of design discourse in test teams
A-2	Reflection and quality of design discourse in control teams
A-3	Regression Curves for Test Teams
A-4	Regression Curves for Control Teams
A-5	How do we handle data requests for external resources?

Abstract

Context: The quality of software design highly depends on the design thinking and the cognitive processes of the designers, as shown in many design studies. One of these cognitive processes is the ability to reflect on one's own design during a design discourse. However, software engineering design methodologies have largely ignored such design thinking process.

Objective: We theorize a two-mind model of design thinking. Mind 1 is about logical design reasoning and Mind 2 is about the reflection on our reasoning ability. The problem solving ability of Mind 1 has often been emphasized in software engineering. The reflective Mind 2, however, has not received much attention. In this study, we want to find out if Mind 2 can improve design discourse, a prerequisite of design quality.

Method: We conducted an exploratory experiment on 12 student groups, divided into test groups and control groups. We provided external reflections to the test groups. No reflections were given to the control groups. We analyzed the quality of the design discourse in both groups.

Results: We found that reflection (Mind 2) improves the quality of design discourse (Mind 1) under certain preconditions. The results highlight the significance of reflection as a mean to improve the quality of design discourse.

Conclusions: We conclude that software designers need both Mind 1 and Mind 2 to obtain a higher quality design discourse, as a foundation for a good design.

0.1 Introduction

The many standards, methodologies and processes have not provided any guarantee for good software design. Software designers ultimately have to make good design decisions. Making good design decisions depends on many factors including a good software design knowledge, a suitable design approach and good problem solving skills. Amongst these factors is the ability of a designer to *reflect* on his/her thinking as well as taking on the feedback by many stakeholders.

Software design is a complex endeavor for many reasons. First, software designers often work in a complex environment and are overloaded with requirements and technical information (Simon, 1973). This situation may restrict designers to what they are able to pay attention to, and leave gaps in their considerations. Second, biases can lead to misrepresentation of facts and arguments (Stacy and MacMillan, 1995; Tang, 2011). Third, designers may inappropriately apply their incomplete knowledge in a new and unfamiliar situation (Epley and Gilovich, 2006).

A general approach to cope with these challenges is to adopt some development processes and standards. This approach is not entirely effective. The issue is that designers can make bad design decisions despite using a development process. Human thinking and reasoning are still crucial components in design thinking. In a study, researchers asked software designers “What are the issues in the decision?” and “What are the options to deal with the issues?”. It was demonstrated that by asking these two simple questions, the quality of user interface design improved significantly for less experienced designers (Tang et al., 2008). This kind of reflective questioning can be an additional intellectual tool that designers may use to improve decision quality and to gain new understanding (Boud, 1985).

Reflection is studied widely in different academic disciplines such as philosophy, education, psychology and professional practice. A common interpretation of reflection is the basic mental process with either a purpose or an outcome or both, that is applied in situations where material is ill-structured or uncertain and where there is no obvious solution (Moon, 1999). Software designers often deal with wicked problems (Rittel and Webber, 1973). In such complex circumstances, it is easy to fall into biases or issues mentioned above. Reflection is a means to allow designers to think at a different level to catch such biases.

We theorize that software design thinking comprises two distinct mental activities, problem solving with a reasoning mindset and reflection with a feedback mindset. We hypothesize that novice software designers can benefit from thinking with these two minds: (a) *reasoning* (Mind 1) where one can systematically reason with a design environment such as requirements, constraints, design problems and solution selection. The software community already has some understanding about this process. For instance, there is a general approach to analyzing, synthesizing and evaluating design solutions (Hofmeister et al., 2005); (b) *reflective thinking* (Mind 2) such as the scrum process provides a feedback mechanism on a design and its reasoning (Talby et al., 2006). Reflective thinking can be used to challenge the way we design software. Quality design thinking and design discourse is a prelude to a quality design (Dorst (2011)). However, there has been no study, as far as we can tell, on how reflections influence the quality of design discourse. In this study, we focus on

reflection as a means to improve software design reasoning.

Section 0.2 provides a background of the various areas involved in this work: design decision making, logical argumentation, quality of design argumentation, problem solving and reflection. In Section 0.3, we describe a two-level software design thinking framework. Mind 1 considers four design reasoning activities: *gathering design contexts*, *formulating design problems*, *exploring design solution options* and *decision making*. Mind 2 is a reflective system to challenge the activities of Mind 1.

We conducted an exploratory experiment to understand how the two design thinking minds work together. We examine if Mind 2 helps to improve design reasoning. The experiments were conducted with Master students in software engineering courses in two universities. Student groups were divided into test and control groups. During each design session, we recorded the students' design discourse and we encoded them to analyze the reflections the students made. We analyzed the relationship between reflections and the quality of design discourse. This is described in Section 0.4.

We report our results in Section 0.5. We found that reflections help to improve the design discourse, and external reflections facilitate the reflection of students. Ultimately, high quality design discourse is a foundation of good design.

0.2 Background: Software Design Thinking and Reasoning

Software design is a tricky business, especially designing highly complex software systems. Design activity is said to be wicked (Rittel and Webber, 1973), chaotic and ill-structured (Simon, 1973). The wickedness in software design is about how complexity contributes to difficulties in design, and how it affects the ways people think about design (Budgen, 2003). As such, we need an understanding in related areas to improve software design thinking (Petre et al., 2010). In this section, we first discuss some of the issues of good decision making. We then explore reflections and design reasoning.

0.2.1 Human Issues

Bad design decisions can be due to many different reasons. Some of these reasons are discussed below.

0.2.1.0.1 Cognitive Biases A cognitive bias is a distortion of judgment due to psychological effects and insufficient regards of probability (Tversky and Kahneman, 1974). Although biases have been studied by psychologists extensively, the effects of biases on software development is relatively unknown. Anecdotal evidence of such phenomenon, however, is plentiful in the software industry. As an example, some programmers constantly underestimate programming complexity and efforts. These programmers are biased about their ability to deliver.

Humans are subject to cognitive biases in general (Kahneman and Tversky, 1972). This can happen during software design activities (Stacy and MacMillan,

1995; Tang, 2011). These biases can cause bad design decisions, and result in poor quality software systems. Decision makers can make decisions based on partial information (Epley and Gilovich, 2006), but as new and relevant information becomes available, decision makers are reluctant to change their initial decisions. This is called **anchoring**. Sometimes a designer's anchor is so entrenched that he would not change a decision even when the decision is obviously contradictory to the new information. In a previous experiment, we have observed that less experienced designers tend *not* to change their design solutions once they are made (Tang et al., 2008). We have also observed that with experienced designers, it is possible to anchor on the first solution that comes to mind (Tang et al., 2010). **Confirmation bias** is a tendency for people to favor information that confirms their own preconception or hypothesis (Oswald and Grosjean, 2004). A recent study has shown that confirmation bias can happen with professionals (Calikli et al., 2010) and academia (Hergovich et al., 2010). Other types of biases occur. They can be caused by faulty memory systems, personality, social attributes, behavioral attributes and so on. These biases may also influence software design decision making.

0.2.1.0.2 Poor quality context and requirements Bad design decision happens partially due to poor quality context and requirements (Paul et al., 2006). The contexts and the requirements should be accurate, relevant, and complete. When context lacks these qualities, the design decisions that are based on them can be faulty. For instance, an architect specifies in a requirement statement that a system should be modifiable. Such a specification is inaccurate because a designer can interpret this in many different ways, which in turn can lead to less than ideal design decisions.

0.2.1.0.3 Overlooking design problems Some designers do not explicitly identify the problems that they need to solve in a design. Instead they think solutions. This may lead to over-simplification of design problems and that the design space is not explored thoroughly. Additionally, design problems are often not knowable before design activities begin, and they are hard to identify because they evolve with the solution (Dorst, 2006).

0.2.1.0.4 Bounded rationality Software designers often deal with new and highly complex issues. There are limitations, called bounded rationality, in terms of how much information can be processed (Simon, 1996). A designer may not be able to process all the necessary relevant information and make multiple and interrelated design decisions at one time. An example is that students can focus only on a few important design criteria at the start of a project (Burge and Brinkman, 2010).

0.2.2 Design Reasoning Approaches

Many design techniques and methodologies have been proposed to analyze and synthesize software design. They have their uses in improving the discipline but few of them address the kind of reasoning issues mentioned above. Zannier et al. (2007) have found that the more structured the problem space is, the more rational a designer would approach a design. Some designers use problem

framing as a strategy to plan design activities and to manage complexity and the interplay of design components. [Hall et al. \(2002\)](#) and [Nuseibeh \(March 2001\)](#) suggest to use problem frames to provide a means of analyzing and decomposing problems, and allowing a designer to iterate between the problem space and the solution space. [Maher et al. \(1996\)](#) suggest that solutions are developed as problems are identified during design explorations. This model suggests that the development of a solution leads to further identification of design problems. Problem-solution co-evolution theory is described by [Dorst and Cross \(2001\)](#). They suggest that creative design is about developing and refining the problem space and the solution space iteratively.

Problem solving requires mental representations of the problems and of the information and knowledge to solve the problems ([Björklund, 2013](#)). All these information need to be framed appropriately for sense making and reasoning. There are many models to represent how designers reason and make decisions. [Langley et al. \(1995\)](#) suggest that decisions can be nested, snowballed or recurring. [Gero and Kannengiesser \(2004\)](#) suggest that a design object is a result of function-behavior-structure in its situated environment. Many have proposed approaches and models for software design rationale. [Zimmermann et al. \(2009\)](#) suggest a model to represent software architecture design decisions. [Tang and van Vliet \(2009b\)](#) suggest a design decision process model by using design context, problem and solution iteration. [Paul et al. \(2006\)](#) describes engineering thinking in terms of the ability to reason critically with the use of quality attributes such as clarity, accuracy, precision, relevance and so on. [Kruchten et al. \(2006\)](#) suggest a model of evolving from a design idea, to a tentative decision, going through different stages of a decision being challenged, approved or rejected. All these methods aim to model and improve design reasoning, with different characteristics and emphasis.

Software design decisions often deal with design elements such as constraints ([Tang and van Vliet, 2009a](#)), assumptions ([Lago and van Vliet, 2005](#)), risks ([Poort and Vliet, 2011](#)), costs, benefits and their tradeoffs ([Kazman et al., 1998](#)). These design elements are essential to design reasoning. There are many techniques that improve the analysis of these software design elements. In this article, we use a general model proposed by [Tang and van Vliet \(2009b\)](#) as a base to describe software design reasoning. This model has a similar form to QOC ([Maclean et al., 1996](#)) and gIBIS ([Conklin and Begeman, 1988](#)) with extra information to support reasoning, and the activities in the model require reflection (see Section [0.3](#)).

0.2.3 Design Reflection

Reflection is a thinking process that challenges and revises one's decision. In this article, we suggest to use reflection systematically during a software design process because it makes designers aware of the potential design reasoning issues (Section [0.2.1](#)). Many have studied reflection as a mental process for the purposes of education (e.g. Dewey, van Manen), of epistemological issues (e.g. Herbamas), and of professional practice (e.g. Schön). Recently, few in software engineering community have addressed reflection (e.g., [Dyba et al. \(2014\)](#)). In what follows we provide an overview of different interpretations on reflection followed by how we interpret reflection in this work.

0.2.3.0.5 Reflection as a deliberative rationality van Manen (1977) argued: “the deliberative rationality of empirical-analytic theory does not offer norms for choosing among alternative practical possibilities. Thus, the rationality of the best choice is defined in accordance with the principles of technological progress-economy, efficiency, and effectiveness. When the nature of this constraint is recognized, the need for a higher level of deliberative rationality becomes apparent”. In sum, he perceives reflection as thinking at a higher and deliberate level, a way to achieve higher rationality.

0.2.3.0.6 Reflection as a conversation about the thinking process

In the reflection-in-action theory, Schön (1983) pointed out that a designer must consider the situations that he is faced with, and reflects on it to obtain new ideas. The synthesis of a solution requires knowledge, both software design knowledge as well as domain knowledge. Schön challenges the reliance on *technical rationality*—design based on rationalizing with concrete technical knowledge—because in real life, problems are often highly complex. Instead, he suggested a *reflective conversation* (or reflection-in-action) through asking relevant questions over the course of formulating a solution. Similarly, McCall suggested that design is a critical conversation between *ideation* and *evaluation* (McCall, 2010), where an evaluation is a form of reflection in which a designer assesses if a design solution can achieve a goal. Valkenburg and Dorst (1998) suggested that design is about move-test-experiment, involving both action and reflection. In their study, the students had to act on a design to discover its feasibility, only then could they reflect on what was and what was not possible in their design. Reymen et al. (2006) defined the process of design reflection as critically considering the designer’s perception of the design situation and of the remembered design activities. Using a structured reflection process, they concluded that designers need reflection early in a design process.

0.2.3.0.7 Reflective software practices Inspired by Schön’s reflection-in-action theory, software engineering community mainly views reflection as conversation about *past experience* and application of the learning in the current problem solving. Dingsoyr et al. (2006) refer to reflection as a learning process to uncover tacit knowledge that exists in architects. They propose reflective questions as a mean to promote reflection. Dyba et al. (2014) view reflection as an *experimental learning process*, during which a designer consciously applies the learning from past experiences. Bull and Whittle (2014) conjecture that reflection should be part of software engineering education. In their approach of teaching reflection to software design students reflection has quite broad connotation covering constant questioning coming from peer students or instructors, collaborative learning, peer review, and critiques.

Babb et al. (2014) perceive reflection quite close to the agile philosophy and treat reflection as practices integrated into the agile software construction process. During communication, agile reflections become relevant for agile management practice and for team reflection where shared goals are useful to generate insights about questions and to discover the “why” part of the problem. Thus, mini-retrospective meetings fit perfectly for agile development practices during daily operations. Retrospective meetings can be seen as reflections in which the team assesses using a neutral facilitator (e.g., a Scrum Master (Schwaber and

Beedle, 2002)). A sort of day-to-day short reflections are useful to adapt and adjust the strategy and assumptions of agile projects in Scrum teams and where the Scrum master exercises leadership and coaches less expert team members enabling team decision-making and increasing local decision-making processes during daily meetings. Lamoreux (2005) encourages reflection as part of the learning cycle in agile practices and how to help teams to improve their reflection meetings overcoming the initial reluctance spending time to reflect regularly. The author mentions some roadblocks to effective reflection planning the sprint meetings and how reflection was more successful when planning meetings did not occur. Talby et al. (2006) analyze the role of reflections in agile teams and the retrospective elements in team reflection.

Agile software development approaches like Extreme Programming (XP) suggest adding a reflective practice based on Schön’s work, where learning is emphasized through reflection. Talby suggests also use reflection to discuss a specific problem, adapt the reflective practice to be “agile” through small incremental improvements (e.g, the reflection cannot exceed one hour), and to encourage high communication and feedback as well. Such reflective technique shows positive results for agile decision-making where reflections play a key role to communicate the rationale behind key practices.

People-centric methods like the agile philosophy is shifting to promote introspection and reflection as common practices to articulate the underlying assumptions in the today’s software design and recognize the value for constantly reframing the problem and its solution (Nerur and Balijepally, 2007). Santos et al. (2013) provide a pattern-language for inter and intra-team knowledge sharing in agile development to increase reflection in collective meetings across agile teams. Others like Babb et al. (2014) feel necessary to streamline the reflective practice and suggest a reflective agile learning model (REALM) which embeds such reflective practice into agile development. The authors examine two reflective practices (i.e., “reflection-in-action” concerning individual responses to problems and “reflection-on-action” concerning post-evaluation activities to validate individual and team questions and address learning spikes) where agile teams can make the tacit assumptions explicit and answer questions like “What did/does he mean?”. Despite some reflective practices in the agile movement, there is no distinction between reflection in reasoning and technical reflection. Additionally, the studies of agile development do not have a comprehensive framework of reflective practices.

0.2.3.0.8 Sources of reflection Lockyer et al. (2004) adopted a model of reflection in which reflection comprises of *external feedback* and *self distillation*. Reflection can be internal emerging from sources such as knowledge, experience, critical incidents, new information, insights and so on. We call this kind of reflection *internal reflection*, as apposed to reflections coming from external sources such as feedback, criticisms, questioning and analogies, called *external reflection*.

0.2.3.0.9 Defining reflection for software design There are many interpretations of reflection as discussed above, especially amongst different domains. In this article, with Schön (1983), we distinguish between reflection at a *technical level* and reflection at a *reasoning process level*. Reasoning reflection challenges

ones' design thinking at a process level which does not consider technical design details. For instance, a reasoning reflective question is to ask if one has considered all possible solution options. An example of a technical design reflection, on the other hand, is to ask if SSL is secure enough. Our focus in this study is to investigate reflection of design reasoning, and not of technical design reflection. Therefore, from now on by reflection we mean reasoning reflection. Moreover, based on Lockyer et al. (2004), we distinguish two types of reflection based on its source: (i) *external reflection* coming from external sources, and (ii) *internal reflection* coming from self experience and understanding.

0.3 Reflection and Reasoning

Software design has often been seen mainly as a problem-solving. This approach is incomplete due to the the lack of considerations of the human issues discussed earlier. Software design also requires a reflective mind that challenges our decisions that are often presumed right. For instance, designers can face unfamiliar design situations in which they have little prior experience, they need to explore areas such as the contexts, the environment, the problem space and the solution space. Their decision process is dynamic (Dorst, 2006). Decision formation requires inspecting these areas from different approaches, including the business approach and the technical approach. Reflection is one way to challenge and encourage the reasoning mind to explore. In addition, designers often have to adapt to a changing environment, which involves how well a designer reflects and changes ones' approach during design. Lastly, novice designers¹ often require help to enhance their problem-solving and reasoning abilities. One known way for such help is to prompt their reflective thinking (Dorst, 2006). Taken together, reflective mind is an essential aspect that so far has not yet received much attention.

In this section, we introduce a design thinking framework that comprises of two minds. Figure 1 depicts the two minds. Mind 1 is about logical argumentation which is based on reasoning and rationale. Recent research in design rationale and reasoning have heralded their use in design (Zimmermann et al., 2009; Singley and Carroll, 1996; Tang et al., 2009). Mind 2 is about reflecting on how Mind 1 thinks. Mind 2 is about conscious questioning and reflection on how we reason with software design. Mind 1 is about the generic design reasoning activities. Mind 1 activities include information gathering, exploring design problems, exploring design solutions, and evaluating and making decision. Mind 1 and Mind 2 thinking are orthogonal to System 1 and System 2 thinking². Both Mind 1 and Mind 2 thinking for novices, the subjects of this study, require a conscious and deliberate act (i.e. System 2 thinking), this is different to System 1 thinking whereby judgments are autonomous (Kahneman, 2011). We suspect that as designers are more experienced with using Mind 1 and Mind 2 thinking, they can become more autonomous with less deliberate efforts.

¹According to Dorst and Reymen (2004) a novice designer is a designer who will consider the objective features of a situation, as they are given by the experts (e.g., tutors or supervisors), and will follow strict rules to deal with the problem.

²System 1 thinking is about autonomous thinking; it is quick and it requires less cognitive load of the thinker. System 2 thinking is about conscious and deliberate act of thinking; it is effortful and it involves conscious decision making (Kahneman, 2011).

As mentioned in Section 0.2.2, there are different methods to reason with a design. We briefly describe them in Section 0.3.1, the details of design reasoning are outside the scope of this paper.

Figure 1: Software Design Thinking Model - Mind 1 and Mind 2

0.3.1 Mind 1: Design Reasoning and Decision Making

In solving a design problem, if a designer is familiar with the problem and the solution, she might find it quite easy to solve that problem relying on her experience. However, if the problem is new and unfamiliar, how would a designer approach it? In such a case, one way is to gather relevant requirements or premises, contemplate what problems to solve, and if a solution is suitable for satisfying a set of requirements and contexts (Tang and van Vliet, 2012). Four general design reasoning activities (Figure 1) are identified:

- *Identifying Relevant Context and Requirements.* This activity is about information gathering, it usually takes place first but could continue until a design is complete. The contexts of a design include factors that affect design decisions (Tang and Lau, 2014). Requirements include functional and non-functional requirements that need to be catered for in a design.
- *Formulating & Structuring Design Problems.* A designer needs to formulate what design issues need to be addressed. The articulation of design problems is a key component in problem-solution co-evolution (Dorst and Cross, 2001; Hall et al., 2002).
- *Creating Solution Options.* Each partial solution addresses some design problems, and potentially creates new design problems. Problems and solutions can co-evolve. When design decisions are made or prior decisions are backtracked, new contexts and problems could surface. The explorations of problems and solutions are described by Maher et al. (1996), Dorst and Cross (2001) and Shaw (2012).
- *Deciding on a Solution.* A designer makes a logical decision by analyzing and evaluating the solution options.

These four activities do not need to be in sequence and they are iterative. Although contexts usually have to be gathered first because the goals and requirements need to be established for any system. However, the contexts are often enriched as new problems and solutions are uncovered. This model does not suggest the order of the activities. The process of applying this model is incremental and iterative. A design decision can spark new design problems, leading to new design options and contexts. A design outcome or decision is the logical conclusion from evaluating design contexts, problems and solutions.

The generic activities in this model show what a software designer would need to do, but software designers often do not consciously think of how to use these steps. There is often no explicit plan or awareness to use these steps, such as exploring problems, or evaluating options using some trade-off analysis. Many studies have shown that some of these steps are not taken. For instance, [Curtis et al. \(1988\)](#) reported that designers often do not have the knowledge or context they need in a design; [Tang et al. \(2008\)](#) showed that designers do not fully explore problem space and design options before committing to a decision. To improve this situation, we need a way to help software designers improve their design reasoning. **Reflection** on how one reasons is a potential way to improve design reasoning and decision making.

0.3.2 Mind 2: Design Reflection

Mind 2 is a questioning mind that is used to challenge and seek feedback. It reflects on how a designer reasons with her design and it questions Mind 1. This is a mechanism which checks for biases, anchoring and omissions. [Schön \(1983\)](#) suggested that practitioners have a situated reflective conversation. A specific or situated problem that a designer faces can spark ideas to create a potential solution. Reflection can help evaluate such ideas. [McCall \(2010\)](#) described this as a critical conversation on ideation and evaluation in which the process of design is the generation of ideas (ideation) interacting with the evaluation of the ideas. The reflection we study is to challenge our technical reasoning during design.

Reflection can be generated by a third-party (externally) or by self-assessment (internally). Self-assessment is to question one's own approach in design. Such questioning can be about many things such as the quality of the context (i.e. what else do I need to know?), one's potential biases in the evaluation of a decision (i.e. is the trade-off in the decision fair?), the accuracy of the requirements (i.e. are there any related requirements?) and so on. There are four key areas of design reflection (Mind 2) as depicted in Figure 1.

- **Reflect on the Contexts and Requirements** - this reflection is used to evaluate if the contexts and requirements identified are relevant, complete and accurate. For instance, we check if all requirements have been accounted for (i.e. completeness); or if certain contexts and requirements are relevant to a design discussion or not (i.e., relevance); or if we have accurately described certain requirements such as system performance (i.e. accuracy).
- **Reflect on Design Problems** - this reflection evaluates if the design problems have been well articulated. [Dorst \(2006\)](#); [Tang and Lau \(2014\)](#)

both found that design problems can be unclear, imprecise and poorly articulated. Reflection is aimed at challenging these situations.

- **Reflect on Design Solutions** - this reflection challenges how design solutions are arrived at. [Tang and Lau \(2014\)](#) found in industrial case studies that the following situations occurred: (a) sometimes designers do not provide solution options even when they exist; (b) design solutions are accepted with no requirements; (c) a solution does not address any requirements. In these cases, reflection helps to evaluate the quality and the appropriateness of a solution.
- **Reflect on Design Decision** - a reflection to evaluate whether a design decision is sound and valid. For instance, a reviewer needs to reflect on the pros and cons of each solution options. Trade-off analysis is one such reflection ([Kazman et al., 1998](#)). Risks ([Poort and van Vliet, 2012](#)), assumptions ([Lago and van Vliet, 2005](#)), constraints ([Berg et al., 2009](#)) can also be evaluated when a decision is made.

Reflection is a quality assurance process that needs to be actionable and explicit. In this study, we provide (external) reflections to student groups by asking reflective questions on their design reasoning. Table 1 contains a list of reflective questions that we used. These questions challenge how a designer thinks design. The questions in Table 1 are based on four generic design reasoning activities (in columns) from the reasoning activities depicted in Figure 1. The rows in Table 1 show the design reasoning techniques. These design reasoning techniques include trade-off analysis, risk analysis, assumption analysis, constraint analysis and problem analysis. As discussed briefly in Section 0.2.2, these techniques are documented in various software design literature. A summary of these techniques is reported in [Tang and Lago \(2010\)](#). Designers need to consider their assumptions, constraints and risks with respect to the design activities. For instance, Mind 2 can challenge if implicit assumptions are made unknowingly on the context of a design; or if there are any risks and constraints that are taken for granted in a solution. Table 1 shows some of the reflective questions that can be asked. We defined these questions with the aim to challenge design reasoning thinking (Mind 1). These questions are by no means complete, rather they provide examples of generic and technically-independent questions.

0.4 Methodology

In this section we describe our research approach, experiment design, experiment execution, and data analysis method.

0.4.1 Research Approach

In this work, we model two mind sets: Mind 1 and Mind 2. We suggest that these two minds interact with each other during software design. Many works have described reasoning processes in design ([Dorst and Cross, 2001](#); [Tang and van Vliet, 2009b](#)) but there is little exploration on how reflections (Mind 2) influence design reasoning. In this study, we postulate that reflections from

Table 1: Reflective questions used by Mind 2 to reflect on Mind 1

		Design Reasoning Activities				
		Design and Requirements	Contexts and Requirements	Design Problems	Design Solutions	Decision Making
Reasoning Techniques	Assumption Analysis	What assumptions are made?	What assumptions are made?	Do the assumptions affect the design problem?	Do the assumptions affect the solution option?	Is an assumption acceptable in a decision?
	Risk Analysis	What are the risks that certain events would happen?	What are the risks that certain events would happen?	How do the risks cause design problems?	How do the risks affect the viability of a solution?	Is the risk of a decision acceptable? What can be done to mitigate the risks?
	Constraint Analysis	What are the constraints imposed by the contexts?	What are the constraints imposed by the contexts?	How do the constraints cause design problems?	How do the constraints limit the solution options?	Can any constraints be relaxed when making a decision?
	Problem Analysis	What are the contexts and the requirements of this system? What does this context mean?	What are the contexts and the requirements of this system? What does this context mean?	What are the design problems? Which are the important problems that need to be solved? What does this problem mean?	What potential solutions can solve this problem?	Are there other problems to follow up in this decision?
Design	Tradeoff Analysis	What contexts can be compromised?	What contexts can be compromised?	Can a problem be framed differently?	What are the solution options? Can a solution option be compromised?	Are the pros and cons of each solution treated fairly? What is an optimal solution after trade-off?

Mind 2 can help Mind 1, and thus improve the quality of design discourse. Our research addresses a specific research question:

RQ) Do reflections (Mind 2) improve the quality of design discourse of novice designers? If so, how?

To answer our research question we carried out an exploratory experiment with Master students of the software engineering program. We chose exploratory experiments as our research method (Franklin, 2005). We had no preconception as to what hypothesis to test. However, we wanted to discover what would happen if we treated a group with external reflections, as compared to a group that received no treatments.

We chose to use students in this experiment because they are not encumbered by how experience may interact with reflective questions. In a previous experiment (Tang et al., 2008), it was found that junior working designers benefited positively from reasoning, whilst experienced designers did not. Some experienced designers found that the right answer came to them naturally because of previous exposures to situations. If an experiment is to study reflections on experienced designers, then the familiarity factor would need to be isolated in a study. In this case, our objective was to study how reflection influence novices, so students as subjects are appropriate (Kitchenham et al., 2002).

This exploratory experiment enabled us to investigate and evaluate the design thinking conditions of novices with and without the treatments. We allocated students into the test groups (with treatment) and the control groups (without treatment). We made external design reflections in our meetings with the test groups. We made no external reflections with the control groups. We then compared the design activities of the two groups. We observed the ef-

fects of external reflections on internal reflections (Mind 2), and we investigated how reflections (both external and internal) influence the quality of a design discourse.

0.4.1.0.10 Reflection Firstly, as mentioned in Section 0.2.3, in this study we focus on reflection at reasoning process level. Accordingly, reflective questions such as those listed in Table 1 challenge how students reason with their design. These challenges do not deal with the actual technical design directly. This allows us to distinguish between reflective thinking (Mind 2) without directly providing technical hints to the test groups. For example, we avoided asking leading questions like “How do we encrypt a communication channel?” or “What are the security principles”, as these questions hint to the students that they need to consider communication channel encryption. Instead, we ask generic questions like “Have you considered the constraints imposed by the requirements?”, or clarification questions such as “What do you mean by security measures?” if the students mentioned the use of security. These questions were considered as generic reasoning reflection, because they challenge the reasoning of the students.

Secondly, our study was designed to examine if and how reflections bring improvements to a design discourse. In this study, we did not examine the relationships between reflection and the quality of a final design. This is because the students cannot reach a final design from one session of discussion, plus there are many other factors that influence a software design. However, one could suggest that the quality of design discourse has a positive impact on the quality of software design. If we show that the test groups make better design decisions, then they have a better chance to end up with a better design. As such, we studied a more immediate outcome, i.e. whether reflections improve design discourse during a practical session with the students.

0.4.1.0.11 Design Discourse Quality The quality of a design discourse is about its *completeness*, *accuracy* and the *relevance* (Paul et al., 2006). When the students discuss a design topic, they reason with a design matter using one of the activities outlined in Figure 1. That is, they gather information, ask questions, contemplate and select solutions. If a discussion topic is well explored, meaning that all the known requirements and constraints relevant to a topic are addressed, then it is considered *complete*. If a design conversation is *relevant*, it means that a designer does not digress to irrelevant or unrelated topics during a design conversation. If the content about a topic is *accurate*, it means that the information is free from error and precise. In the exploratory experiment, we codify the quality of the design discourse as it progressed. Through this process, we examine if the design discourse had progressed and improved towards the goal of the design.

0.4.2 Designing the Exploratory Experiment

To design and report the experiment we followed the guidelines proposed by Wohlin et al. (2012). The exploratory experiments were conducted in two universities, in the Netherlands, VU University Amsterdam (VU), and in Spain, Rey Juan Carlos University (URJC). The setup of the experiment was identical in both

universities: it took place in a practical session with each student team; the practical session was about the design phase of the broader course project, and it was one hour long. This session, from student's perspective, was about receiving feedback on the way they tackle the design problem of their project. In both test and control teams the lecturers³ were available to answer technical questions of students. Only test groups, however, received reflective questions during the sessions from the lecturers. In total, we had 7 test groups and 5 control groups. Before the experiments students had a lecture on design decision making using QOC (Maclean et al., 1996). We also provided the students the template proposed in (Gu et al., 2010) to document their design decisions. An example of this template is provided in Table A-5. In what follows we further explain the details of our experiment design in terms of experiment context, goals and variables, subjects, objects, design, and instrumentation. s

0.4.2.0.12 Context The context of the experiment is a practical session of the Service Oriented Design course in both universities. All the students were at the Master's level and they were all taught design reasoning tactic before the session. The domain of the course projects were different in the two universities: at VU e-government services that government offers to citizens, such as registering a new-born child, and at URJC Traffic Control Services.

0.4.2.0.13 Goal and Variables Our study was designed to examine if and how much improvements reflections bring to a design discourse. The independent variables are the *external reflections*. The dependent variable is the *quality of the design discourse*. As noted the quality is about the completeness, accuracy and the relevance of a design discourse (Paul et al., 2006). To measure the quality of the design discourse we use five-level coding as shown in Table 2. The five-level coding provides an evaluation about how well the students reason during their design discourse.

0.4.2.0.14 Subjects The subjects are student groups of the Service Oriented Design course in both universities. The groups are randomly assigned to test and control groups. In a group setup, members verbalize their design and so it is easier to evaluate their design discourse, compared with using a think-aloud process for individuals.

0.4.2.0.15 Objects The experiment object is the design discourse in the practical session.

0.4.2.0.16 Experiment Design In our experiment we used the following design tactics suggested by Wohlin et al. (2012).

- *Randomization.* The student teams are randomly assigned to be a test group or a control team. The teams are formed randomly as well.
- *Blocking.* No systematic approach to blocking is applied. The decision to randomly select student teams with different backgrounds, rather than looking at teams with similar education background can be viewed as providing blocking for difference between the backgrounds.

³The lecturers are the authors of this paper.

0.4.2.0.17 Instrumentation To guide lecturers' performing the experiment, we prepared instruments for data collection and experiment execution. To collect data, we video/audio recorded and transcribed all the students sessions. In addition, to guide the lecturers in providing external reflections, a checklist of reflective questions (see Table 1) was provided.

0.4.3 Experiment Execution

0.4.3.0.18 Preparation The students were not aware of what aspects were going to be studied. They were informed that the lecturer were there to help them in the design phase of their projects. The students, from their point of view, did not primarily participate in an experiment, they were attending a practical session of the course. Related to theory, the students were trained on architectural decision-making and the techniques that improve the analysis of various design options using QOC. We did not teach or inform the students about the quality of design discourse reflection. Prior to the practice session, the students had delivered one assignment on their envisioned business services. The practice session was the first time they were exploring their design issues and design options.

0.4.3.0.19 Execution The experiment was executed over the course of two days in the two universities. Each session was 1 hour. The experiment had not been allowed to affect the course objectives, the only difference was that some student groups received reflective questions and the others did not. Out of the 12 participating teams, 7 were in the test group and 5 teams were in the control group. The experiment was, as stated earlier, run within Service Oriented Design course and in university environments. In the beginning of the session we explained: "this is a practice session in which you should start thinking about your design issues and design options; we are mainly here to observe, and possibly answer your questions". In this way, we framed the students to consider the session as a design exercise.

0.4.4 Data Analysis

To investigate the effects reflections have on the quality of design discourse, we analyzed each of the practical session transcripts. We chose transcript coding (Miles and Huberman, 1994) as our qualitative analysis method. To carry out the coding systematically, we devised the coding procedure. By codifying the transcripts, we mapped each of the practical sessions on the framework presented in Section 0.3. This way we were able to distinguish the two minds (design reasoning and reflection). Furthermore, we evaluated the quality of design discourse using the five-level coding as shown in Table 2.

0.4.4.1 Codes

For creating the codes we followed the suggestion of Miles and Huberman (1994), to have an initial set of codes, called 'start-list' that is refined during the analysis. Our start-list stems from the elements of the framework represented in Figure 1 as well as the research question of this study.

As shown in Table 2, there are 3 categories of codes: (i) *design reasoning*, (ii) *reflection*, and (iii) *design discourse quality scales*. The design reasoning reflects the four different *reasoning activities* (see Figure 1). *Reflection* is grouped by either *external reflection* (i.e. reflection from an external party) or *internal reflection* (i.e. self-reflection). Finally, the *design quality measures* represent whether design reasoning is accurate, relevant and complete. The five-level coding provides an evaluation in which to support how well the students reason during their design discourse.

Table 2: List of codes

Category	Code	Name	Description
Design reasoning (Mind 1)	PR	context and requirement	Identify contexts and requirements of a design
	DI	design issue	Addressing design problems
	DO	design option	Identifying possible solution options
	DD	design decision	Evaluating solution options to make a decision
Reflection (Mind 2)	ER	external reflection	Reflection induced by an external source
	IR	internal reflection	Reflective thought from internal awareness
Design discourse quality scales	L5	level 5	All of completeness, accuracy and relevance are present, with rationale to support design reasoning
	L4	level 4	All of completeness, accuracy and relevance are present
	L3	level 3	Two of completeness, accuracy or relevance are present
	L2	level 2	One of completeness, accuracy or relevance is present
	L1	level 1	Reasoning is totally incomplete, inaccurate and irrelevant

0.4.4.2 Coding Procedure

Each transcript was independently codified by two researchers. Disagreements were resolved by a third researcher. To make the codification systematic, we devised the following coding procedure:

- **Step 1: Transcribe the audio / video.** We transcribe the time and the discussions of each session.
- **Step 2: Codify the reasoning activities.** This step codifies the design reasoning activities observed in students' design discourse based on the transcript. For example, in line 6 of Transcript 1 (see Table 4) the students were addressing a design problem: "*How to make sure the data of students remain confidential?*". Therefore, we codified this argument as a design issue (DI). In this way, we were able to map each team's design discourse on Mind 1 activities (see Figure 1).
- **Step 3: Codify design reflections.** This step codifies the external and internal reflections. To identify the external reflections we used three criteria: (i) it is a reflective question posed by the lecturer, (ii) it challenges the student's reasoning of a design, and (iii) it does not involve the technical details of a design directly. For example, Transcript 1, line 1 meets all

the three criteria, thus it is an external reflection (ER). Internal reflection (IR) is similar to external reflection, the only difference is that students themselves challenge their own reasoning.

- **Step 4: Grade the quality of the design discourse.** Considering the description of each quality measure in Table 2, we measured the quality of the design discourse. To keep the evaluation as simple (and robust) as possible, there was no complicated rubric for evaluating design discourse; rather, we used a simple five-point scale based on the three qualities. We felt that a more granular scale would lead to overprecision (Haran et al., 2010). As shown in Table 2, each scale indicates the number of qualities that are present where an L1 indicates none of the qualities are present and an L5 represent all the qualities are present, with rationale to support design reasoning.

0.4.4.3 Coding Consistency

After encoding each transcript separately by two researchers, we checked the consistency of the encoding of the reflections and the quality. There were 104 codified instances of reflection. The Cohen's Kappa Coefficient (Cohen, 1968) was 0.616, which indicated substantial agreement between the two coders. We checked the level of agreement on the quality of the design discourse using the Intraclass Correlation Coefficient (Donner and Koval, 1980). With the 151 observations obtained from the transcripts, we obtained an average measure of 0.852, which indicated there was substantial agreement between the two coders. These measures provided assurance that the interpretations of the reflection and the quality of design discourse were consistent among the coders.

0.4.4.4 Qualitative Causality Analysis

In this study we were interested to know if and how reflection relates to the quality of design discourse. We wanted to explore how students talk design and how they react to reflections in a design discourse. For that, we needed to link the conversation that happened in the practice session with interpretations. Therefore, we used a *qualitative causality analysis* method to *explore* and *explain* the effects of reflection on the quality of design discourse. For causality, we follow the definition of Miles and Huberman (1994), i.e., *what led to what over a period of time*.

We chose qualitative studies because, with Lincoln and Guba (1985), we consider qualitative to be more effective compared to quantitative methods in studying this kind of issues. With a close up look, qualitative methods have the advantage to show what preceded what, through retrospection. We used the *explanatory effect matrix* (Miles and Huberman, 1994), as the display to support seeing “what led to what”.

An explanatory effect matrix displays a set of events (here reflection and design discourse) and traces the consequential outcomes (here quality of design discourse). For the rows of matrix, we put “reflections”—things that might change the quality of the design discourse. For the columns, we included “Meaning of reflection for students” and “how students responded”. Table 3 shows this matrix. Looking through the coded transcripts, we filled-in the matrix and then used this matrix to draw conclusions. By moving row by row, we were able to

analyze the effects of each external reflection. By looking down the columns, we were able to generalize the effects of reflection on quality of design discourse.

Table 3: Explanatory effect matrix of Transcript 1

Reflection	Interpretation of reflection	How students responded
External Reflection: What are your design issues for this business service?	To prompt the students to reflect on the design issues	The student answered the question by pointing out security as an issue, and tried to justify why it was an issue.
External Reflection: What does confidentiality mean for your domain?	The lecturer sought further clarification of the context to see if the students understood why confidentiality was relevant in this situation	The answer given was superficial and no further information was given in this answer
External Reflection: Can you rephrase this issue in form of a question.	The lecturer made another external reflection asking the student to phrase the design issues in form of a question.	One student added data confidentiality as a design issue, providing more specific issues to be addressed. Another student added user authorization as a design issue, a separate security issue but also specific to security. <i>Here, we see that the two external reflections helped the students to improve the quality design argumentation. Interestingly, the external reflection of the lecturer had triggered students to find other design issues.</i>

0.5 Results

We plotted the design reasoning (i.e., Mind 1) and reflections (i.e., Mind 2) over the observed period of about an hour each session. The resulting graphs (shown in Table A-1 and Table A-2) depict the reflections and the quality of the discussions over the design discourse. An example graph is shown in Figure 2 illustrating the design discourse quality over time. The y-axis shows the quality levels from level 1 (lowest) to level 5 (highest). The quality level of a design discourse was based on the coding outlined in Table 2. The x-axis indicates time. The labels on the graph illustrate the quality of one of the context and requirements (● labeled ‘PR’), design issues (▲ labeled ‘DI’), design options (■ labeled ‘DO’), or design decisions (+ labeled ‘DD’). The events below the x-axis shows the reflections that were made by the lecturer (◆ i.e., external reflection) or the students themselves (★ i.e., internal reflection).

There are many curves in a graph, and each curve represents a conversation about a certain design topic. For example, the conversation presented in Transcript 1 (see Table 4) is visualized in the dashed area in Figure 2. First, the curve represents that the students talked about design issues. Second, it indicates that three external reflections occurred in this conversation. Finally, an upward curve indicates improvements in the accuracy of their argumentation.

This conversation illustrates how external reflections affect the quality of a design discourse. As an answer to external reflection by the lecturer (ER8 in Figure 2 i.e., line 1 in Transcript 1), the student raised the issue of security (line 2). This issue, however, was vague. The student used the word “security”

Figure 2: Design discourse and reflections and in T1

vaguely without a clear notion of what it means to their system. By asking how the students defined confidentiality, the lecturer reflected on the design issue (ER9 i.e., line 3 Transcript 1); in answering this question the issue became more articulated (in line 4) (improving the quality to level 4). Interestingly, the external reflection of the lecturer had triggered students to find other design issues (line 7).

For readability reasons, we have folded the explanatory effect matrix into the transcript interpretation (see Table 4) to present the results of analysis in the chronological order of the conversation. The *interpretation* column contains how the reflections have occurred, what prompted the internal reflections, the reactions of the students, and the interpretation of their design discourse.

Table 4: Transcript 1 Interpretation

Design Dialogue	Interpretation
<p>1. Lecturer What are your design issues for this business service?</p> <p>2. Student 3 The most important one is security, because we need the data to be confidential.</p> <p>3. Lecturer What does confidentiality mean for your domain?</p> <p>4. Student 3 It means that the data of student is only visible to authorized people.</p> <p>5. Lecturer Can you rephrase this issue in form of a question.</p> <p>6. Student 3 How to make sure the data of students remain confidential?</p> <p>7. Student 1 We also have another issue about authorization Let's say "how to authorize admission staff?"</p>	<ul style="list-style-type: none"> - This was an external reflection to prompt the students to reflect on the design issues - The student answered the question by pointing out security as an issue, and tried to justify why it was an issue - The lecturer sought further clarification of the context to see if the students understood why confidentiality was relevant in this situation - The answer given was superficial and no further information was given in this answer - The lecturer made another external reflection asking the student to phrase the design issues in form of a question. - One student added data confidentiality as a design issue, providing more specific issues to be addressed. - Another student added user authorization as a design issue, a separate security issue but also specific to security. In these two lines, we see that the two external reflections helped the students to think about their approach to a design. It allowed the students to frame their design problems from a generic confidentiality issue to specific design issues that needed to be solved, and led to a more focused design discussion.

From the graphs, we analyzed how reflections influence the quality of design reasoning. We contrast the influence between the test groups and the control

groups. In the analysis, we also draw on evidence from the explanatory effect of the transcripts. In summary, we have three findings, they are discussed in the following sections.

0.5.1 Finding 1 - External Reflection Induces Internal Reflection

We observed that test teams frequently make internal reflections, more so than the control teams.

This is evident from Table 5 that shows teams that had internal reflections during a design discourse. All test teams had generated internal reflections, but only two control teams had generated their internal reflections. The test teams had external reflections put to them, this appeared to have allowed the test teams to self-question during design.

Table 5: Teams that had internal reflections.

	Internal Reflection	No Internal Reflection
Test Team	7	0
Control Team	2	3

Table 6 shows that the number of internal reflections is considerably higher in the test teams as compared to the control teams. Moreover, through the explanatory effect matrix we found various instances of internal reflections that were induced, or inspired, by external reflections. In this regard, we observed that students tend to internally reflect, after external reflections were used to challenge their design. We have counted 16 internal reflections (out of the 22 in total) that have been directly prompted by an external reflection.

As an example, we select Transcript 2 to illustrate how an external reflection induced internal reflections in a design session. Table 7 shows the simplified form of the explanation matrix for this transcript. The subject of the conversation was securing data transmission.

The lecturer in Transcript 2 started by asking what assumptions are made in the business domain (line 1). Student 1 explained that there was an assumption on disability (line 2, Mind 1 speaking). Student 2 agreed with this assumption by saying “OK”, but then Student 2 went on to question what design issue would

Table 6: Number of reflections by test teams and control teams

Test Team	ID	No. of external reflections	No. internal reflections
Test Team	T1	10	4
	T2	13	4
	T3	14	1
	T4	6	6
	T5	11	7
	T6	8	4
	T7	11	2
Control Team	C1	0	0
	C2	0	0
	C3	0	2
	C4	0	1
	C5	0	0

Table 7: Transcript 2 Interpretation

Design Dialogue	Interpretation
<p>1. Lecturer For your business domain, what are the assumptions?</p> <p>2. Student 1 Well, our system must be usable for people with disabilities. Given that we have chosen to use a client developed in-house, we can create it from the scratch to be as usable for disabled people as possible.</p> <p>3. Student 2 OK, so what is our design issue now; is it how to design user interface in such a way that it is usable by disabled people?</p> <p>4. Student 1 Yes, maybe how do we ensure easy access to users with special needs?</p> <p>5. Student 3 An option could be simply use of color, shapes, presence of animations in UI that makes the use easier for disabled people. Another option could be that UI will be integrated with assistive technologies that support people with disabilities. For example speech recognition or screen reader for blind users.</p> <p>6. Student 1 OK, these two options are good, what are our criteria for selecting them? We need to assess them based on ease of use for disabled people and maybe cost?</p> <p>7. Student 3 I think the second option should be selected because of the public nature of our system. Our system should be accessible to a wide range of disabled people, including blind. The first option does not really support blind people. Are there any other options?</p>	<ul style="list-style-type: none"> - An external reflection of the context - A description of what the system was about. - From the external reflection on the context and a description of the context, this student reflected on what design issues and requirements need to be addressed. This student sought further clarifications. - Confirmation that the requirements need to be further explored. - This student explored potential solution options. - This student made another internal reflection to define the criteria for selecting the design options. This reflection is about the requirements of the system, the student reflected that in order to choose a solution option, he had to know what criteria or basis for the selection. - This student applied the criteria to different groups of users to see if anything was amiss. This was an exploration of the system contexts.

come with this assumption (line 3, internal reflection 1). Student 1 further reflected on “how to ensure a design provides easy access to users with special needs” (line 4, internal reflection 2). This statement is a refinement of line 3 by generalizing that special needs is a more accurate term than disability, and easy access is a more accurate term than designing user interface. Student 3 provided two options for the solution space (line 5, Mind 1 speaking). Student 1 reflected on the criteria to select from these two options (line 6, internal reflection 3). In this example, we can see that the external reflection triggered the discussions and three internal reflections were brought forward. Interleaving the reflections were the Mind 1 exploration of the design context, problem and solution spaces.

0.5.2 Finding 2 - Reflection Improves Quality of Design Discourse

With reflections, the quality of design discourse improves under certain conditions. From the transcripts, we observed that some students tend to describe their contexts and requirements loosely. Such poor design argumentation led to poor quality design discourse. We use examples from transcript 3 and 4 to illustrate how inaccurate contexts and requirements led to low quality design argumentation.

Table 8: Transcript 3

Design Dialogue
<p>1. Student 1 Let's select one of our business services. Maybe "admission management"?</p> <p>2. Student 2 Yes admission management has to be online. Everybody should be able to access it from anywhere and anytime. Because we have international students. So we need a web service.</p>

The students sought to ground the contexts and requirements using the admission business service as example (Transcript 3, line 1). Student 2 argued that the online admission was necessary and shortly afterward drew the conclusion that web services were needed. First, many assumptions were left implicit (e.g., the relationship between international students and the need for online admission). This left incomplete arguments because it depends on what information is to be exchanged during the online admission process. Second, their solution (i.e., Web Services) was based on the belief that international students need it. This argument was not substantiated with facts. This introduced inaccuracy in a design discourse.

We further observed that inaccuracy in problem formulation hampers logical design argumentation too. Transcript 4 illustrates how inaccurate problem formulation leads to insufficient or irrelevant solutions.

Table 9: Transcript 4

Design Dialogue
<p>1. Student 1 Our goal is to make e-Services for chamber of commerce in a way the new companies can easily register and carry out the bureaucratic processes. The goal is to make this process, easier, faster and cheaper.</p> <p>2. Lecturer What is the problem you want to solve?</p> <p>Student 3 In the process of registering a company we have to make sure that it is not a dummy company.</p> <p>Student 1 Maybe a solution is to ask for a digital signature, so we authenticate.</p>

Student 1 clarified the main purpose of their e-Service (Transcript 4, line 1). After providing some generic context the lecturer asked about the problem formulation (line 2). Considering the context provided in line 1 the answer to this question was irrelevant (line 3). Many assumptions that backup the importance of security were left implicit and incomplete. In addition, the problem formulation was inaccurate. For instance, it was unclear what they meant by security measures or why dummy companies had to be identified. Finally, student 1 suggested the 'digital signature' as a solution option for this problem (line 5). Not surprisingly, this options did not sufficiently address the problem.

In this instance, we saw that the design argumentation was imperfect. We could see that the argumentation was not carefully constructed, and there was insufficient design reasoning coming from Mind 1. It seems that the design argumentation was from some intuitions, and the analysis from those intuition was lacking. If such was the case, we wanted to see how reflections or Mind 2 can help to improve the situation.

0.5.2.1 Finding 2.1 - External reflection improves quality

To examine *how much improvements reflections bring to a design discourse*, we need to capture the progression of quality in a design discourse, with and without reflections.

We explored the quality progressions using regression analysis to visualize trends. It should be noted that the regression analysis was used to show trends and not for identifying the causality between reflection and quality. Using regression analysis, the quality of the design discourse of each team was plotted to show its progression as a trend curve (red curve in Figure 3). In constructing the trend curves, we used logarithmic regression as it best fits the design discourse quality curves (the gray curve in Figure 3).

Figure 3: Quality of design discourse in T1

The resulting trend curves, shown in Table A-3 and Table A-4 in the Appendix, revealed the following interesting observation. There is a considerable improvement in the quality of design discourse in test teams. The only outliers were T3 and T7—not showing any quality improvements—for which we describe our explanation in Section 0.5.3. In contrast, the regression curves of control teams, shown in Table A-4, revealed that the quality of design discourse remains constant, over time. We understand that the increase in the quality of design discourse in design teams could be explained as simply designers need time to get to a certain level of quality in their design discourse. However, comparing the progression of the quality in the test and control teams, we noticed that the improvements are more obvious in the test teams and not in the control teams.

To examine *how reflections improve the quality of the design discourse*, we analyzed the explanatory effect matrices. By moving across the explanatory effect matrices, we have found evidence to support that *external reflection helps to improve the quality of design discourse*:

- External reflection challenged the students to be *clearer* and more *precise* with their design discourse. By asking questions like ‘what is the assumption here?’ we made students reflect on what they did not know and what they did not consider to be a relevant context in their design discourse. Likewise, questions such as ‘what is your definition of quality?’, or ‘can you rephrase the issue in terms of a question?’ forced students to think and frame their requirements and argumentation more precisely.

- External reflection triggered students to eliminate irrelevant context, problems and solutions. By asking questions like “does this assumption affect the design problem?” we made students aware of noise in their argumentation.
- External reflection improved *completeness* by prompting them to consider requirements or design arguments that would have been missed. By asking questions like ‘Is this the only option?’ we made students to explore the problem and solution options more thoroughly.

High quality design discourse is a foundation for good design, by constantly reflecting on the various aspects of software design, students are more likely to have a higher quality design discussion. Design discussions appear to be a central mechanism in the development of a common language in design teams. Something that is often thought to be a useful indicator of good design (Lloyd, 2000). Additionally, it was found that design discussions facilitate the mutual understanding of design evaluation criteria which speeds up the design process by allowing designers to find a set of possible designs (Nickerson and Yu, 2013). Besides, we know from (Tang et al., 2010) that an ineffective design discourse can lead to an inferior outcome.

0.5.2.2 Finding 2.2 - Without internal reflection, design quality does not propel

We found that the teams that made *more internal reflections achieved better design discourse*. For instance, the regression curves of T1, T2, T4, T5, T6—teams with 4 internal reflections or more—level out between quality level 3 and 4 (see Table A-3). In contrast, when the number of internal reflections is small, we observed that the accuracy improvements do not propel. For example, in the graph shown in Figure 4 the students received many external reflections, they however had only one internal reflection. This accuracy curve reveals that although external reflections have induced some immediate improvements, the quality of design discourse levels out between level 1 and level 2, and as such does not propel. A similar pattern is evident in the regression curves of control teams C1, C3, C4, which also have few internal reflections. Thus, it suggests that *if reflection is not internalized by a designer to a point that they start to reflect on the design discourse themselves, the quality of a design argumentation does not continue to improve*.

0.5.3 Finding 3 - Reflection Cannot Improve Quality without Domain Knowledge

By comparing the different teams, we observed that reflection improves relevance only when the students have some domain knowledge. By asking questions like *why is this an issue?* or *who cares about this problem?* we prompted students to reflect on their contexts and arguments. However, despite whether in a control or test team, if the students do not have the basic knowledge about the domain, they cannot bring relevant information to the design discussions.

Let us consider Transcript 5 where students had some idea about the domain. The students were considering a system to keep track of the history of

Figure 4: Design discourse and reflection in T3

unemployed people. An external reflection was given to prompt why the students wanted to keep track of a status. Apart from the status, other relevant information should be captured. Student 2 suggested to keep relevant information such as training and success rate, hence showing some knowledge about this area. When the students have some domain knowledge about the contexts or the problems, then reflections help them to improve the relevance of a design discussion.

Table 10: Transcript 5 Interpretation

Design Dialogue	Interpretation
<p>1. Student 1 We have to keep the status of jobseekers throughout time. We need to know when they were employed and when unemployed.</p> <p>2. Lecturer Why is their status important for you?</p> <p>3. Student 1 Their status is important because we need to check if the trainings helped them to find the job.</p> <p>4. Student 2 Then we need to keep the history of their training and the success rate of those training.</p>	<p>- Requirements suggested by a student.</p> <p>- An external reflection to seek justifications and contexts of the design argument.</p> <p>- The student suggested that training affects job finding. This interpretation relates skills to the success of finding a job.</p> <p>- This student took the training and success rate as causally related, and further suggested to keep the history for analysis. The statement confirmed and concluded that training history is required.</p>

On the other hand, students that did not have relevant domain knowledge struggled to have a relevant design discussion. Consider Transcript 6, where the conversation was about the design of the data storage component. The students started discussing whether they should use a cloud provider, without considering what needed to be stored and transferred. At this stage, this issue of selecting a cloud provider was irrelevant to the design. Despite an external reflection, students continued their design discussions on cloud storage. Relevant issues would have been about efficiency, security and reliability of the data component. The students needed to understand the domain of data storage, and the relevance of those issues in order to carry out a meaningful design discussion.

When we contrast the quality of the design discourse between the teams, we

Table 11: Transcript 6 Interpretation

Design Dialogue	Interpretation
<p>1. Student 1 We should decide which cloud provider to use for keeping the big data of X-Ray or MRI machines.</p> <p>2. Lecturer But you said you are going to just use the data storage components of legacy systems.</p> <p>3. Student 1 Yes but someday we have to move to the cloud, because this is what government wants.</p> <p>4. Student 2 Well, not in my country, specially the medical data that is private, but let's assume Netherlands mandates use of clouds.</p>	<p>- Student stated the solution, i.e. cloud provider, without any justifications.</p> <p>- The lecturer reflected on the contradiction to an earlier statement.</p> <p>- The student justified this by another requirement (i.e. what government wants) without a proper rationale or factual support.</p> <p>- This student pointed out that there were requirement differences amongst countries, but they didn't really know what the government policy was on data storage of sensitive information on the cloud. The discussions did not focus on the storage requirements.</p>

see that external reflections cannot bring the students to discuss more relevant topics if the students lacked the basic domain knowledge. It appears that if the students do not have the knowledge to consider relevant issues (as in Transcript 6), then they cannot improve their design discourse. Therefore, *only when the domain knowledge is sufficient would reflections improve the quality of a design discourse.*

0.6 Limitations

Lincoln and Guba (1985) argue that qualitative research should be objective, reliable, and internally and externally valid. Below, we discuss the limitations of this work and how we addressed them.

0.6.0.0.1 Objectivity Objectivity ensures that the conclusions depend on the subjects and the conditions of the case rather than on the researcher (**Miles and Huberman, 1994**). The emphasis here is on the replicability of a study by other researchers. One threat in this regard is that the external reflection was posed by the lecturers. Thus, the students might craft their responses to show a more positive picture of “how they design”. To minimize the researcher bias, we followed the suggestion of **Miles and Huberman (1994)** and adopted the following tactics. First, we informed students that they were not assessed based on the practice session and the lecturers were there to discuss their projects with them. Second, some of the researchers were teaching assistants helping students with their projects and did not have a role in evaluating the students; this allowed us not to be seen as assessors and minimized the biases leading students to crafting their responses. Accordingly, we consider the threats related to researchers’ effect to be limited.

0.6.0.0.2 Reliability The underlying issue of reliability is whether the process of study is consistent over time across researchers and methods (**Miles and Huberman, 1994**).

A risk here is that the ranking of the quality of design discourse is subjective. To mitigate this risk, for each transcript two researchers, independently, codified and prepared the rankings. Then, the codifications and rankings were compared, and in case of disagreements consensus was achieved through the third researcher. As noted in Section 0.4.4 the Cohen's Kappa Coefficient (Cohen, 1968) was 0.616, which indicated substantial agreement between the two coders. We checked the level of agreement on the quality of the design discourse using the Intraclass Correlation Coefficient (Donner and Koval, 1980). With the 151 observations obtained from the transcripts, we obtained an average measure of 0.852, which indicated there was substantial agreement between the two coders. These tactics and measures helped us ensure that the study is done with desirable quality.

0.6.0.0.3 Internal Validity Internal validity aims at ensuring that the collected data enables researchers to draw valid conclusions. We conducted an exploratory experiment to understand the effects of reflections on design discourse. The results provide strong indication that there are differences between having and not having external reflections. The improvements in design discourse quality, the induced internal reflections and the trends are evidence that differentiate the two groups. However, we note the limitations of the study that there are explanatory factors we cannot isolate in the experiments. These limitations include students' background, students' experience, and their reflective abilities.

Another limitation is that the design thinking we observed was based on dialog from audio recording. Non verbalized exchanges such as pointing and looks could not be observed. We assume that in a group discussion, most of the considerations were communicated verbally.

This study is also subject to selection risk. The experiment was a mandatory part of the service oriented design course, thus the selection of subjects is not random, which involves a threat to the validity of the experiment. We mitigated this risk by random allocation of students to teams. The students were also told that their grading in the course was not depending on the performance in the experiment, only on their serious attendance. There is also the risk that the students lack motivation; they may consider their participation as a waste of time. The lecturers, however, have made a strong effort in motivating the students.

Finally, in this work, we recognize the difficulties in segregating Mind 1 and Mind 2 activities, and their causal relationships. We addressed this risk by (i) scoping down Mind 2 to a set of reflective questions, and (ii) systematically analyzing the causality between Mind 1 and Mind 2, through zooming into what led to what over each practice session.

0.6.0.0.4 External validity External validity defines to what extent findings from the study can be generalized. A possible threat to external validity of our study is having students as subject population. What mitigates this threat is that we made no assumptions about the students' experience or educational background and teams consisted of a random mix of novice and more experienced students. In another study, reflective questions such as "What are the

issues in the decision?” and “What are the options to deal with the issues?” have shown to have an impact on design quality (Tang et al., 2008). This work provides more evidence to support the theory.

However, this threat is reduced by using Masters students which, if not already working in industry, are close to finalize their education and start working in industry. In addition, the setting is intended to resemble a real design situation, in a meeting room with whiteboard and flip-charts. The assignments are real-world cases directly proposed by industrial practice. In sum, as our subjects, the assignment and the settings resembles real cases to a reasonable extent, the results can be of interest to software design teams that include novice designers. Moreover, our findings support the findings by Sonnentag (1998) in that she found that high performers spent twice as much time on feedback processing than moderate performers. We found that more reflections, or feedback, help to improve the quality of design discourse.

0.7 Discussion

There are many definitions of reflection. We selected a narrow interpretation so that we can differentiate the act of reflection and the act of reasoning in our exploratory experiment. It allowed us to inject reflections without injecting knowledge to the test teams, thus we avoided providing extra information and leading questions that would bias our experimental results. The main result of this study has indicated that a reflective mind can help novice designers to improve design discourse. This has implications on how to educate and train novice software designers. We also observe a number of issues.

0.7.0.0.5 Reflection Inertia Our study suggests that the students, as novice designers, have the natural tendency to limit their design explorations to their initial ideas when investigating the context, problem and solution spaces. The *Law of Least Effort* states that people will gravitate to the least demanding course of effort to accomplish the same goals (Kahneman, 2011). This can happen despite teaching design reasoning in courses partly because the practices are not readily instilled into students. This laziness or inertia may be one of the reasons why novice designers do not explore the design spaces sufficiently. External reflective questions appear to prompt Mind 1 to explore more of the design spaces and overcome this inertia. This indicates that the habit of reflection needs to be taught and practiced.

0.7.0.0.6 Need for a Reflection Advocate Given that external reflection makes a difference to the quality of design discourse, it appears to be worthwhile to inject reflective questions in the software design teams to challenge decision making. This is somewhat different to the devil’s advocate role in agile software teams (McAvoy and Butler, 2009) in that reflection questions and challenges design reasoning. The role of a reflection advocate is to ask questions similar to that in Table 1 to check the assumptions, risks, problem statements and other areas of decision making. In our work we acted in a similar form like Scrum’s masters in the role as facilitators with the students to bring reflections and introspection explicit. However, we found that bringing reflections explicitly helped the students to reframe the problem and the solution architecture.

0.7.0.0.7 Reflection, Reasoning and Intuition System 1 thinking is autonomous. Experienced software designers dealing with familiar domains can think autonomously and arrive to a solution directly. The reasoning in this situation is proven, and there is no need to redo the reasoning every time when the same problem arises. However, when a designer is facing new and complex issues, such solution-driven strategy may not work very well. A software designer may need a scoping-questioning strategy in order to “get a better feel” for what the problem is about (Tang and van Vliet, 2012). Reflective design thinking is deliberate and it challenges the autonomous thinking one might easily take.

0.7.0.0.8 Reflection in Real-world Practice The study of reflection in real-world environment is more difficult, as experience, self-belief and other factors can be difficult to isolate in the experimental results. One research approach is the use of case study as it was done by Schön (1983) and many others. In a study, multiple designers were given the same problem, and their design sessions were recorded. Researchers observed and analyzed their design discourse (Petre et al., 2010). We intend to study experienced designers in a similar way but with reflections a stimulus.

0.8 Conclusion

Software design has often been considered as a problem solving exercise. Recent design studies have indicated that behavioral, cognitive and other factors also influence design thinking. Software design is a complex business involving many human decisions, but software engineering design methodologies have not seriously considered how these other factors influence design reasoning and the resulting design. One of these factors is the ability of a software designer to reflect on one’s design.

We theorize that one way to approach software design thinking is to model design reasoning in two Minds. Mind 1 is the reasoning mind that focuses on the process of logical design reasoning. This is a model which outlines key design reasoning activities (Mind 1). Mind 2 is the reflective mind that challenges Mind 1 by asking reflective questions. It checks if a novice designer has approached a design logically and adequately. We classified two types of reflections: (i) *external reflections* were made by a third party to designers; (ii) *internal reflections* were challenges self-raised by designers. In this study, the test groups were given external reflections, and the control groups received no external reflections.

We setup an exploratory experiment to study how reflection influences design reasoning in twelve groups of students. We have three findings. First, external reflections triggered internal reflections. Without external reflections, less than the half of the student teams in the control group raised any internal design reflections. Second, both types of reflections helped designers to improve the quality of design discourse; without internal reflection, external reflection itself did not improve the quality. Our evidence shows improvements in the quality of design discourse in a trend as reflections were raised. Third, the quality of the design discourse improved only when students had basic domain knowledge. Students could not identify the relevant contexts and design issues in their discussions if they lacked knowledge about the subject domain. Our findings are limited to novice designers, we are yet to understand how experienced de-

signers may react to reflective questions. These findings show the importance of reflections in design reasoning to novice designers, and the necessity of having the two minds working together. The results imply that reflective practice has its place in education. From a practical software engineering perspective, we are also concerned with how reflective questions can help improve the process and the quality of software design. Our next step is to experiment with experienced designers. This paper provides early results to show that asking reflective questions during software design is valuable.

Bibliography

- Babb, J., Hoda, R., Norbjerg, J., 2014. Embedding reflection and learning into agile software development. *IEEE Software* 31 (4), 51–57.
- Berg, M. v. d., Tang, A., Farenhorst, R., 2009. A constraint-oriented approach to software architecture design. In: *Proceedings of the Quality Software International Conference (QSIC 2009)*. pp. 396–405.
- Björklund, T. A., 2013. Initial mental representations of design problems: Differences between experts and novices. *Design Studies* 34 (2), 135–160.
- Boud, D., 1985. *Reflection: Turning experience into learning*. Routledge.
- Budgen, D., 2003. *Software design*. Pearson Education.
- Bull, C. N., Whittle, J., 2014. Supporting reflective practice in software engineering education through a studio-based approach. *IEEE Software* 31 (4), 44–50.
- Burge, J. E., Brinkman, B., 2010. Using rationale to assist student cognitive and intellectual development. *Human Technology* 6 (1).
- Calikli, G., Bener, A., Arslan, B., 2010. An analysis of the effects of company culture, education and experience on confirmation bias levels of software developers and testers. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2. ICSE '10*. ACM, New York, NY, USA, pp. 187–190.
- Cohen, J., 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin* 70 (4), 213.
- Conklin, J., Begeman, M., 1988. gIBIS: A hypertext tool for exploratory policy discussion. In: *Proceedings ACM Conference on Computer-Supported Cooperative Work*. pp. 140–152.
- Curtis, B., Krasner, H., Iscoe, N., 1988. A field study of the software design process for large systems. *Commun. ACM* 31 (11), 1268–1287.
- Dingsoyr, T., Lago, P., van Vliet, H., sep 2006. Rationale promotes learning about architectural knowledge. In: *8th International Workshop on Learning Software Organizations (LSO)*. Rio de Janeiro, Brazil.
- Donner, A., Koval, J. J., 1980. The estimation of intraclass correlation in the analysis of family data. *Biometrics* 36 (1), 19–25.

- Dorst, K., 2006. Design problems and design paradoxes. *Design issues* 22 (3), 4–17.
- Dorst, K., 2011. The core of ‘design thinking’ and its application. *Design Studies* 32 (6), 521 – 532, interpreting Design Thinking.
URL <http://www.sciencedirect.com/science/article/pii/S0142694X11000603>
- Dorst, K., Cross, N., 2001. Creativity in the design space: co-evolution of problem-solution. *Design Studies* 22 (5), 425–437.
- Dorst, K., Reymen, I., 2004. Levels of expertise in design education. In: Lloyd, P., Roozenburg, N., McMahon, C., Brodhurst, L. (Eds.), *Proceedings of the 2nd International Engineering and Product Design Education Conference*.
URL <http://doc.utwente.nl/58083/>
- Dyba, T., Maiden, N., Glass, R., 2014. The reflective software engineer: Reflective practice. *IEEE Software* 31 (4), 32–36.
- Epley, N., Gilovich, T., 2006. The anchoring-and-adjustment heuristic. *Psychological Science* 17 (4), 311.
- Franklin, L., 2005. Exploratory experiments. *Philosophy of Science* 72 (5), 888–899.
- Gero, J. S., Kannengiesser, U., 2004. *Design studies* 25 (4), 373–391.
- Gu, Q., Lago, P., van Vliet, H., Sept 2010. A template for soa design decision making in an educational setting. In: *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*. pp. 175–182.
- Hall, J., Jackson, M., Laney, R., Nuseibeh, B., Rapanotti, L., 2002. Relating Software Requirements and Architectures Using Problem Frames. In: *IEEE Joint International Conference on Requirements Engineering*. pp. 137–144.
- Haran, U., Moore, D. A., Morewedge, C. K., 2010. A simple remedy for over-precision in judgment. *Judgment and Decision Making* 5 (7), 467–476.
- Hergovich, A., Schott, R., Burger, C., 2010. Biased evaluation of abstracts depending on topic and conclusion: Further evidence of a confirmation bias within scientific psychology. *Current Psychology* 29, 188–209.
- Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., America, P., 2005. Generalizing a model of software architecture design from five industrial approaches. In: *Fifth Working IEEE / IFIP Conference on Software Architecture (WICSA 2005)*. pp. 77–88.
- Kahneman, D., 2011. *Thinking, fast and slow*. Penguin.
- Kahneman, D., Tversky, A., 1972. Subjective probability: A judgment of representativeness. *Cognitive Psychology* 3 (3), 430–454.
- Kazman, R., Klein, M. H., Barbacci, M., Longstaff, T. A., Lipson, H. F., Carrière, S. J., 1998. The architecture tradeoff analysis method. In: *ICECCS*. pp. 68–78.

- Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K., Rosenberg, J., 2002. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on* 28 (8), 721–734.
- Kruchten, P., Lago, P., Vliet., H. v., 2006. Building up and reasoning about architectural knowledge. In: *Quality of Software Architecture (QoSA)*. Vol. LNCS 4214. Springer-Verlag, pp. 43–58.
- Lago, P., van Vliet, H., 2005. Explicit assumptions enrich architectural models. In: *Proceedings 27th International Conference on Software Engineering (ICSE)*. pp. 206–214.
- Lamoreux, M., 2005. Improving agile team learning by improving team reflections [agile software development]. In: *Agile Conference, 2005. Proceedings. IEEE*, pp. 139–144.
- Langley, A., Mintzberg, H., Pitcher, P., Posada, E., Saint-Macary, J., 1995. Opening up decision making: The view from the black stool. *organization Science* 6 (3), 260–279.
- Lincoln, Y. S., Guba, E. G., 1985. *Naturalistic Inquiry*. Vol. 75. Sage Publications.
- Lloyd, P., 2000. Storytelling and the development of discourse in the engineering design process. *Design Studies* 21 (4), 357 – 373.
- Lockyer, J., Gondocz, S. T., Thivierge, R. L., 2004. Knowledge translation: The role and place of practice reflection. *Journal of Continuing Education in the Health Professions* 24 (1), 50–56.
- Maclean, A., Young, R., Bellotti, V., Moran, T., 1996. Questions, Options and Criteria: Elements of Design Space Analysis. In: Moran, T., Carroll, J. (Eds.), *Design Rationale: Concepts, Techniques and Use*. Lawrence Erlbaum Associates, Ch. 3, pp. 53–106.
- Maher, M. L., Poon, J., Boulanger, S., 1996. Formalising design exploration as co-evolution: A combined gene approach. Tech. rep., University of Sydney.
- McAvoy, J., Butler, T., 2009. The role of project management in ineffective decision making within agile software development projects. *European Journal of Information Systems* 18 (4), 372–383.
- McCall, R., 2010. Critical conversations: Feedback as a stimulus to a creativity in software design. *Human Technology* 6 (1), 11–37.
- Miles, M. B., Huberman, A. M., 1994. *Qualitative data analysis: An expanded sourcebook*. Sage.
- Moon, J. A., 1999. *Reflection in learning and professional development: Theory and practice*. Psychology Press.
- Nerur, S., Baliyepally, V., 2007. Theoretical reflections on agile development methodologies. *Communications of the ACM* 50 (3), 79–83.

- Nickerson, J. V., Yu, L., 2013. Going meta-design space and evaluation space in software design. In: Petre, M., Hoek, A. V. D. (Eds.), *Software Designers in Action: A Human-centric Look at Design Work*. Chapman & Hall/CRC, Ch. 19.
- Nuseibeh, B., March 2001. Weaving Together Requirements and Architecture. *IEEE Computer* 34 (3), 115–119.
- Oswald, M. E., Grosjean, S., 2004. Confirmation bias. Psychology Press, Hove, pp. 77–96.
- Paul, R., Niewoehner, R., Elder, L., 2006. *The thinker's guide to engineering reasoning*. Foundation for Critical Thinking, Dillon Beach, CA.
- Petre, M., van der Hoek, A., Baker, A., 2010. Editorial. *Design Studies* 31 (6), 533 – 544, special Issue Studying Professional Software Design.
- Poort, E. R., van Vliet, H., 2012. Rcda: Architecting as a risk- and cost management discipline. *Journal of Systems and Software* 85 (9), 1995–2013.
- Poort, E. R., Vliet, H. v., 2011. Architecting as a risk- and cost management discipline. In: *Proceedings of the Ninth IEEE/IFIP Working Conference on Software Architecture*. pp. 2–11.
- Reymen, I., Hammer, D., Kroes, P., Van Aken, J., Dorst, C., Bax, M., Basten, T., 2006. A domain-independent descriptive design model and its application to structured reflection on design processes. *Research in Engineering Design* 16 (4), 147–173.
- Rittel, H. W. J., Webber, M. M., 1973. Dilemmas in a general theory of planning. *Policy Sciences* 4 (2), 155–169.
- Santos, V., Goldman, A., Guerra, E., Cleidson, D. S., Sharp, H., 2013. A pattern language for inter-team knowledge sharing in agile software development, 2–21.
- Schön, D. A., 1983. *The reflective practitioner : how professionals think in action*. Nueva York, EUA : Basic Books.
- Schwaber, K., Beedle, M., 2002. *Agile software development with scrum*.
- Shaw, M., 2012. The role of design spaces. *Software, IEEE* 29 (1), 46–50.
- Simon, H., 1996. *The Sciences of the Artificial*. MIT Press.
- Simon, H. A., 1973. The structure of ill structured problems. *Artificial Intelligence* 4 (3-4), 181–201.
- Singley, M., Carroll, J., 1996. Synthesis by Analysis: Five Modes of Reasoning That Guide Design. In: Moran, T., Carroll, J. (Eds.), *Design Rationale: Concepts, Techniques and Use*. Lawrence Erlbaum Associates, Ch. 8, pp. 241–265.
- Sonnentag, S., 1998. Expertise in professional software design: A process study. *Journal of Applied Psychology* 83 (5), 703–715.

- Stacy, W., MacMillan, J., June 1995. Cognitive bias in software engineering. *Commun. ACM* 38, 57–63.
- Talby, D., Hazzan, O., Dubinsky, Y., Keren, A., 2006. Reflections on reflection in agile software development. In: *Agile Conference, 2006*. IEEE, pp. 11–pp.
- Tang, A., 2011. Software designers, are you biased? In: *Proceeding of the 6th international workshop on SHaring and Reusing architectural Knowledge. SHARK '11*. ACM, New York, NY, USA, pp. 1–8.
- Tang, A., Aleti, A., Burge, J., van Vliet, H., 2010. What makes software design effective? *Design Studies* 31 (6), 614–640.
- 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 (v1.4). Tech. rep., Swinburne University of Technology.
- Tang, A., Lau, M. F., 2014. Software architecture review by association. *Journal of Systems and Software* 88 (0), 87–101.
- Tang, A., Tran, M. H., Han, J., van Vliet, H., 2008. Design reasoning improves software design quality. In: *4th International Conference on the Quality of Software-Architectures (QoSA)*. pp. 28–42.
- Tang, A., van Vliet, H., 2009a. Modeling constraints improves software architecture design reasoning. In: *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*. IEEE, pp. 253–256.
- Tang, A., van Vliet, H., 2009b. Software Architecture Design Reasoning. In: Babar, M. A., Dingsi, T., Lago, P., van Vliet, H. (Eds.), *Software Architecture Knowledge Management*. Springer Berlin Heidelberg, Ch. 9, pp. 155–174.
- Tang, A., van Vliet, H., 2012. Design strategy and software design effectiveness. *IEEE Software* Jan-Feb, 51–55.
- Tversky, A., Kahneman, D., 1974. Judgment under uncertainty: Heuristics and biases. *Science* 185 (4157), 1124–1131.
- Valkenburg, R., Dorst, K., 1998. The reflective practice of design teams. *Design studies* 19 (3), 249–271.
- van Manen, M., 1977. Linking ways of knowing with ways of being practical. *Curriculum inquiry* 6 (3), 205–228.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A., 2012. *Experimentation in software engineering*. Springer.
- 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 (6), 637 – 653, qualitative Software Engineering Research.

Zimmermann, O., Koehler, J., Leymann, F., Polley, R., Schuster, N., 2009. Managing architectural decision models with dependency relations, integrity constraints, and production rules. *Journal of Systems and Software* 82 (8), 1249–1267.

AppendixA Diagrams

Table A-1: Reflection and quality of design discourse in test teams

T4

T5

T6

T7

Table A-2: Reflection and quality of design discourse in control teams

Table A-3: Regression Curves for Test Teams

Table A-4: Regression Curves for Control Teams

Table A-5: How do we handle data requests for external resources?

Design issue	D6: How do we handle data requests for external resources?
Context	The E-Health system is a service-based system which often need to use external resources to retrieve certain data that could not be supplied by services internal to the system. E.g. To retrieve information about new medications the E-Health system will need to request pharmaceutical services provided by external service providers. Requesting data from an external resource, outside the internal network, requires additional communication. An effective way of handing these external data request is needed.
Quality attributes	Cr1: nfr_03_Consistency, Cr2: nfr_02_Performance
Identifier	D6-Opt1: sending request when needed.
Description	The system always requests to external partners whenever data is needed.
Relationship(s)-	
Status	Rejected
Evaluation	Cr1: This option ensures maximal consistency, because all data are queued on-the-fly from the data provider. Cr2: Always queuing data providers over the network could result in very bad performance of the whole system, because it needs higher network bandwidth (more data is transferred) and also data providers would have to process more requests.
Architectural options	
Rationale	This option is rejected because in the context of the E-Health system, the same types of data (such as information about medicines) is often requested by different doctors. Most of the data provided by external providers are relatively static, which means the data does not change frequently (such as the description of medicines, profiles of medical experts). Therefore data consistency is not a big issue. However, large number of requests for same type of data is regarded as unnecessary traffic that hinders performance.
Identifier	D6-Opt2: Caching requested data.
Description	Data that has been requested within certain time interval is cached.
Relationships	D7 (What is the most appropriate time interval for caching different types of data?) D8 (What types of data can (and cannot) be cached)
Status	Accepted
Evaluation	Cr1: This option could cause data inconsistency, because data might change while the system is in use. Cr2: This option reduces network traffic and consequently increases performance.
Rationale	For the data that is relatively static, data inconsistency (Cr1) would not be a big problem. For the data that dynamically changes (such as the storage of medicines, availability of devices from partner hospitals), properly configured time intervals for caching different types of data can improve data consistency.