In two minds: how reflections influence software design thinking

Maryam Razavian\textsuperscript{1,2,*}, Antony Tang\textsuperscript{3}, Rafael Capilla\textsuperscript{4} and Patricia Lago\textsuperscript{1}

\textsuperscript{1}Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
\textsuperscript{2}School of Industrial Engineering, Eindhoven University of Technology, The Netherlands
\textsuperscript{3}Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia
\textsuperscript{4}Computer Science Department, Universidad Rey Juan Carlos, Spain

ABSTRACT
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 and judgments. 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, or reflection, can improve design discourse, a prerequisite of design quality. We conducted multiple case studies with 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. 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. 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. Copyright © 2016 John Wiley & Sons, Ltd.

1. INTRODUCTION

The many standards, methodologies and processes available to the software designer 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 [1]. Amongst these factors is the ability of a designer to reflect on his/her thinking as well as taking on the feedback from 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 [2]. This situation may restrict designers in what they are able to pay attention to, and leave gaps in their considerations. Second, biases can lead to misrepresentation of facts and arguments [3, 4]. Third, designers may inappropriately apply their incomplete knowledge in a new and unfamiliar situation [5].

A general approach to cope with these challenges is to adopt some development processes and standards. This general 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 one 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 [6]. This kind of reflective questioning can be an additional intellectual tool that designers may use to improve decision quality and to gain new understanding [7].

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 [8]. Software designers often deal with wicked problems [9]. In such complex circumstances, the design process is easily biased by the factors discussed above. Reflection is a means to allow designers to think at a different level to catch such biases.

In an earlier study [6], it was found that junior designers tend to anchor on those solutions that first come to mind. If students were asked about their design rationale, the students backtracked their design, otherwise, they continued designing despite having design issues. With this observation, we conjecture that junior or novice designers do not systematically reflect on what enters into their minds during design. They lack the abilities needed to systematically sort and filter information that is important. In a recent study, it was found that most professionals and students have satisficing behavior, meaning that they do not analyze thoroughly before making design judgments [10]. 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 the following two minds:

- **reasoning** (Mind 1) where one can systematically reason with a design context 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 [11].
- **reflective thinking** (Mind 2) is comparable to the way that the Scrum process uses a feedback mechanism on a design and its reasoning [12]. 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 [13]. However, there has been no study, as far as we can tell, on how reflections can influence the quality of design discourse. In this study, we focus on reflection as a means to improve software design reasoning.

Section 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 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 used to challenge the activities of Mind 1.

We conducted multiple case studies to understand how the two design thinking minds work together. We examine if Mind 2 helps to improve design reasoning. The study was conducted with Masters 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 4.

We report our results in Section 5. We found that reflections help to improve design discourse, and external reflections, i.e., reflections coming from external sources such as feedback, facilitate the reflection of students. Ultimately, high quality design discourse is a foundation of good design.

## 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 [9], chaotic and ill-structured [2]. The wickedness in software design is about how complexity contributes to difficulties in design, and how it affects the ways people think about design [14]. As such, we need an understanding of the related areas to improve software design.
Bad design decisions can be because of many different reasons. Some of these reasons are discussed below.

2.1.1. Cognitive biases. A cognitive bias is a distortion of judgment because of psychological effects and insufficient regard of probability [16]. Although bias has been extensively studied by psychologists, 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 effort. These programmers are biased about their ability to deliver.

Humans are subject to cognitive biases in general [17]. This can happen during software design activities [3, 4]. These biases can cause bad design decisions, and result in poor quality software systems. Decision makers can make decisions based on partial information [5], 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 or she 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 [7]. We have also observed that even with experienced designers, it is possible to anchor on the first solution that comes to mind [18].

Confirmation bias is a tendency for people to favor information that confirms their own preconception or hypothesis [19]. A recent study has shown that confirmation bias can happen with professionals [20] and in academia [21]. 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.

2.1.2. Poor quality context and requirements. Bad design decision happens partially because of poor quality context and requirements [22]. Knowledge about the context and the requirements should be accurate, relevant and complete. When these qualities are lacking, the design decisions that are based on them can be faulty. For instance, an architect may specify in a requirement statement that a system should be modifiable. Such a specification is imprecise because a designer can interpret this in many different ways, which in turn can lead to less than ideal design decisions.

2.1.3. Overlooking design problems. Some designers do not explicitly identify the problems that they need to solve in a design. Instead they think about solutions. This may lead to over-simplification of design problems so 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 [23].

2.2.4. 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 [24]. 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 [25]. It was also found that software designers sometimes make satisficed judgments, i.e. they perform some reasoning and consider this good enough to make judgments [11].

2.2.5. Satisficing behavior. Satisficing indicates that a decision maker makes a decision that is good enough because he cannot completely explore the design spaces [24]. Zannier et al. [26] found that designers have satisficing behavior and do not necessarily always try to explore exhaustively. In an interview, one of their participants said that ‘... if I’m quite confident that I know of a good solution that does what I need I will just do that and that’s a question of experience … I know that I can do four things here, I don’t know where any of those four things will lead me. So let’s just see
what happens. . .’. Tang and van Vliet [11] found in their study that most software designers give few reasons before they make design judgments.

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. [26] have found that the more structured the problem space is, the more rationally 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. [27] and Nuseibeh [28] suggest using 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. [29] 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 [30]. 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 [31]. All this information needs to be framed appropriately for sense making and reasoning. There are many models to represent how designers reason and make decisions. Langley et al. [32] suggest that decisions can be nested, snowballed or recurring. Gero and Kannengiesser [33] 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. [34] suggest a model to represent software architecture design decisions. Tang and van Vliet [35] suggest a design decision process model by using design context, problem and solution iteration. Paul et al. [22] 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. [36] 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 of 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 [37], assumptions [38], costs, benefits and their tradeoffs [40]. 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 [35] as a base to describe software design reasoning. This model has a similar form to QOC [41] and gIBIS [42] with extra information to support reasoning, and the activities in the model require reflection (see Section 3).

2.3. Design reflection

Reflection is a thinking process that challenges and revises one’s decision. In this article, we suggest it is desirable to use reflection systematically during a software design process because it makes designers aware of the potential design reasoning issues (Section 2.1). Many have studied reflection as a mental process for the purposes of education (e.g. van Manen [43]), of epistemological issues (e.g. Habermas [44]), and of professional practice (e.g. Schön [45]).

Recently, a few members of the software engineering community have addressed reflection (e.g., Dyba et al. [46]). In what follows we provide an overview of different interpretations on reflection followed by how we interpret reflection in this work.

2.3.1. Reflection as a deliberative rationality. van Manen [43] 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, and a way to achieve higher rationality.
2.3.2. Reflection as a conversation about the thinking process. In the reflection-in-action theory, Schön [45] pointed out that a designer must consider the situations that he or she is faced with, and reflect on it to obtain new ideas. The synthesis of a solution requires knowledge, both the 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 [47], where an evaluation is a form of reflection in which a designer assesses if a design solution can achieve a goal. Valkenburg and Dorst [48] suggested that design is about move-test-experiment, involving both action and reflection. In their study, the participants 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. [49] 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.

2.3.3. Reflective software practices. Inspired by Schön’s reflection-in-action theory, the software engineering community mainly views reflection as conversation about past experience and application of the learning in the current problem solving. In the traditional software design methods, design review challenges if anything is amiss. Dingsoyr et al. [50] 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. In Dyba et al. [46] reflection is viewed as an experimental learning process, during which a designer consciously applies the learning from past experiences. Bull and Whittle [51] 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. [52] perceive reflection as being 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 [53]). Day-to-day short reflections are useful to adapt and adjust the strategy and assumptions of agile projects in Scrum teams and one 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 [54] encourages reflection as part of the learning cycle in agile practices and how to help teams to improve their reflection meetings, so overcoming the initial reluctance to spend 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. [12] 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 et al. [12] suggest also using reflection to discuss a specific problem, adapting the reflective practice to be ‘agile’ through small incremental improvements (e.g., the reflection cannot exceed 1 h), 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 approaches like the agile philosophy encourage use of 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 [55]. Santos et al. [56] 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. [52] feel it
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, these make no distinction between reflection in reasoning and technical reflection. Additionally, the studies of agile development do not have a comprehensive framework of reflective practices.

2.3.4. Sources of reflection. Lockyer et al. [57] adopted a model of reflection in which reflection comprises external feedback and self distillation. Reflection can be internal thoughts emerging from sources such as knowledge, experience, critical incidents, new information, insights and so on. We call this kind of reflection internal reflection, as opposed to reflections coming from external sources such as feedback, criticisms, questioning and analogies, called external reflection.

2.3.5. Defining reflection for software design. There are many interpretations of reflection as discussed above, especially amongst different domains. In this article, with Schön [45], we distinguish between reflection at a technical level and reflection at a reasoning process level. Reasoning reflection challenges one’s design thinking at a process level which does not consider technical design details. For instance, a reasoning reflective question is one that asks if one has considered all possible solution options. An example of a technical design reflection, on the other hand, is one that asks 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. [57], 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.

3. REFLECTION AND REASONING

Software design has often been seen mainly as a problem-solving activity. This perspective is incomplete because of the lack of consideration 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 and they need to explore aspects such as the context, the environment, the problem space and the solution space. Their decision process is dynamic [23]. 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 one’s approach during design. Last, novice designers often require help to enhance their problem-solving and reasoning abilities. One known way to provide such help is to prompt their reflective thinking [23]. 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 [34]; [58]; [59]. Mind 2 is about reflecting on how Mind 1 thinks. Mind 2 is concerned about the questioning and reflection of how we reason. Mind 1 is a design mindset involving design reasoning. 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 thinking. 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 [60]. We suspect that as designers gain experience with using Mind 1 and Mind 2 thinking, they can become more autonomous with less deliberate efforts. As mentioned in Section 2.2, there are different methods to reason about a design.
We briefly describe them in Section 3.1, the details of design reasoning are outside the scope of this paper.

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 context [61]. 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 context of a design includes factors that affect design decisions [62]. Requirements include functional and non-functional requirements that need to be catered for in a design.
- Formulating and 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 [30]; [27].
- 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 context and problems could surface. The explorations of problems and solutions are described by Maher et al. [26], Dorst and Cross [30] and Shaw [63].
- 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. Context usually has to be gathered first because the goals and requirements need to be established for any system. However, the context is 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 context. A design outcome or decision is the logical conclusion from evaluating design context, 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. [64] reported that designers often do not have the knowledge or context they need in a design; Tang et al. [6] 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.

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 [45] 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 [47] 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 undertake 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 their strategy 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 Context and Requirements**—this reflection is used to evaluate if the context 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 context 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 [23]; Tang and Lau [62] 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 [62] found in industrial case studies that the following situations occurred: (a) sometimes designers do not consider 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 [40]. Risks [65], assumptions [38], constraints [66] can also be evaluated when a decision is made.

The four types of reflection are interconnected and are carried out iteratively. Reflecting on one aspect of design reasoning activity could spark reflection on other aspects. For instance, when reflecting on a design solution, a designer might realize a shortcoming in the way context and requirements are identified.

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 I contains a list of reflective questions that we used. These questions challenge how a designer thinks about design. The questions in Table I are based on four generic design reasoning activities (in columns) from the reasoning activities depicted in Figure 1. The rows in Table I 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 2.2, these techniques are documented in various software design literature.
<table>
<thead>
<tr>
<th>Design reasoning techniques</th>
<th>Design contexts and requirements</th>
<th>Design problems</th>
<th>Design solutions</th>
<th>Decision making</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumption analysis</td>
<td>What assumptions are made?</td>
<td>Do the assumptions affect the design problem?</td>
<td>Do the assumptions affect the solution options?</td>
<td>Is an assumption acceptable in a decision?</td>
</tr>
<tr>
<td>Risk analysis</td>
<td>What are the risks that certain events would happen?</td>
<td>How do the risks cause design problems?</td>
<td>How do the risks affect the viability of a solution?</td>
<td>Is the risk of a decision acceptable?</td>
</tr>
<tr>
<td>Constraint analysis</td>
<td>What are the constraints imposed by the context?</td>
<td>How do the constraints cause design problems?</td>
<td>How do the constraints limit the solution options?</td>
<td>Can any constraints be relaxed when making a decision?</td>
</tr>
<tr>
<td>Problem analysis</td>
<td>What are the context and the requirements of this system? What does this context mean?</td>
<td>What are the design problems? Which need to be solved? What does this problem mean?</td>
<td>What potential solutions can solve this problem?</td>
<td>Are there other problems to follow up in this decision?</td>
</tr>
<tr>
<td>Tradeoff analysis</td>
<td>What context can be compromised?</td>
<td>Can a problem be framed differently?</td>
<td>What are the solution options? Can a solution option be compromised?</td>
<td>Are the pros and cons of each solution treated fairly? What is an optimal solution after tradeoff?</td>
</tr>
</tbody>
</table>
summary of these techniques is reported in Tang and Lago [67]. Designers need to consider their assumptions, constraints and risks with respect to the design activities. For instance, Mind 2 can challenge whether 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 I 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.

4. METHODOLOGY

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

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 [30]; [35] but there is little exploration on how reflections (Mind 2) influence design reasoning. In this study, we postulate that reflections from Mind 2 can help Mind 1, and thus improve the quality of design discourse. Our research addresses a specific research question:

RQ) How can inclusion of reflections (Mind 2) improve the quality of design discourse of novice designers?

To answer our research question we carried out an exploratory study with the Master students of the software engineering program. We chose a case study approach as our research method [68]. 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 study because they are not encumbered by how experience may interact with reflective questions. In a previous experiment [6], it was found that novice designers benefited positively from reasoning, while experienced designers did not. Some experienced designers found that the right answer came to them naturally because of previous exposures to situations. If a study is to examine 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 [69].

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

4.1.1. Reflection. First, as mentioned in Section 2.3, in this study we focus on reflection at reasoning process level. Accordingly, reflective questions such as those listed in Table I challenge how students reason about 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 asked 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.
Second, 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.

4.1.2. Design discourse quality. The quality of a design discourse is about its completeness, accuracy and the relevance [22]. When the students discuss a design topic, they reason about a design issue 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 it is precise. In this study, we codify the quality of the design discourse as it progressed. Through this process, we examine if the design discourse had progressed and improved toward the goal of the design.

4.2. Case study design

To design and report the case study we followed the guidelines proposed by Runeson and Höst [70]. The case studies were conducted in two universities, in the Netherlands, Vrije Universiteit Amsterdam (VUA), and in Spain, Rey Juan Carlos University (URJC). The setup of the cases was identical in both universities: it took place in a practical session with each student team of four students; the practical session was about the design phase of the broader course project, and it was 1 h 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 lecturers 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 practical sessions the students had a lecture on design decision making using QOC [41]. We also provided the students with the template proposed in [71] to document their design decisions. An example of this template is provided in Table A-5.

Our study satisfies Yin’s criteria of multiple case studies [68], as: (i) it copes with the technically distinctive situation in which there are many more variables than data points—in our case studies, the presence or absence of reflection is the main variable that we examine even though other confounding variables may be present; (ii) it relies on multiple sources of evidence, with data needing to converge—we triangulate our results by having multiple cases in test and control groups. (iii) it is based on pre-established theory—in our study, the conceptual model is the Two Minds theory involving the reflective mind and the reasoning mind; (iv) the boundary between the studied object and its environment is blurred—as design thinking is an internalized mental activity, one can only observe the effect of design discourse without clearly able to identify that a subject is in reflection mode.

In what follows we further explain the details of our case study design in terms of context, goal, case and subject selection.

4.2.1. Context. The context of our case study 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 tactics before the session. The domain of the course projects were different in the two universities: at VUA e-government services that government offers to citizens, such as registering a new-born child, and at URJC Traffic Control Services.

§ The lecturers are the authors of this paper.
4.2.2. **Goal.** Our study was designed to examine if and how much improvement reflections bring to a design discourse. We used a multiple case studies design. Each session with the students was an individual case study. 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 [22]. To measure the quality of the design discourse we used a five-level coding scheme as shown in Table II. The five-level coding provides an evaluation about how well the students reason during their design discourse.

4.2.3. **Case and subject selection.** As shown in Figure 2 we have two groups of cases: Test and Control. This case selection was meant to realize two types of replication suggested by Yin [68], i.e., theoretical replication and literal replication. The test cases are theoretical replication of control cases as they are selected to predict contrasting results because of involvement of reflection. The cases within test and control groups are literally replicated so that they predict similar results. The subjects are student groups of the Service Oriented Design course in both universities. Seven groups were available in VUA and five groups in URJC. There were 4 students in each group. As the number of groups is odd in both universities, we split the groups between test and control such that we had 4 test and 3 control at VUA and 3 test and 2 control at URJC. The student groups were randomly assigned to test and control groups. In a group setup, members need to verbalize their design and so it is easier to evaluate their design discourse, compared with using a think-aloud process for individuals. The unit of analysis is the design discourse in the practical session.

### Table II. List of codes.

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

![Figure 2. Design of multiple case studies.](image-url)
For selection of our subjects we used the following design tactics suggested by Wohlin et al. [72].

- **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.

### 4.3. Case study execution

#### 4.3.1. Instrumentation

To guide the lecturers performing the case studies, we prepared the instruments for data collection and practice session execution. To collect the 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 I) was provided.

#### 4.3.2. Preparation

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 a case study, they were attending a practical session of the course. This was because full disclosure of our research goal and procedures would have compromised the conduct of the case studies as such. If the students knew our research goal, they might have crafted their responses to reflective questions accordingly. Therefore, we followed the suggestion of Runeson and Höst [70] and provided partial disclosure: the goals of practice session was introduced in a higher level of abstraction. After the practice sessions a feedback session was organized where we informed the students about the nature of the case studies. Furthermore students were assured that any information they shared with researchers remained confidential.

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.

#### 4.3.3. Execution

The case studies were executed over the course of two days in the two universities. Each session was 1 h. The case studies were not 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, seven were in the test group and five teams were in the control group. The cases were, 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 encouraged the students to consider the session as a design exercise.

### 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 [73] as our qualitative analysis method. To carry out a coding systematically, we devised a coding procedure. In codifying the transcripts, we mapped each of the practical sessions on the framework presented in Section 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 II.

#### 4.4.1. Codes

For creating the codes we followed the suggestion of Miles and Huberman [73], 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 II, 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 of how well the students reason during their design discourse.

4.4.2. Coding procedure. Each transcript was separately 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 IV) 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 about 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 II, we measured the quality of the design discourse. To keep the evaluation as simple (and robust) as possible, there was no complicated model 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 [74]. As shown in Table II, 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.

4.4.3. Coding consistency. After encoding each transcript separately using 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 [75] 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 [76]. 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.

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 about 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 [73], i.e., what led to what over a period of time.

We chose qualitative studies because, with Lincoln and Guba [77], we consider qualitative to be more effective when compared to quantitative methods in studying this kind of issues. With a close up look, qualitative methods are able to show what preceded what, through retrospection. We used the explanatory effect matrix [73], 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 the 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 III 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.
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 A1 and Table A2) depict the reflections and the quality of the discussions over the design discourse. An example graph is shown in Figure 3 illustrating the design discourse of T1\textsuperscript{1} 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 II. The x-axis indicates time. The labels on the graph illustrate the quality of one of the context and requirements (\textbullet labeled ‘PR’), design issues (\textDelta labeled ‘DI’), design options (\textbullet labeled ‘DO’) or design decisions (+ labeled ‘DD’). The events below the x-axis shows the reflections that were made by the lecturer (\textbullet i.e., external reflection) or the students themselves (\textstar 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 IV) is visualized in the dashed area in Figure 3. 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 3 i.e., line 1 in Transcript 1 in Table IV), the

![Figure 3. Design discourse and reflections in T1.](image-url)
student raised the issue of security (line 2). This issue, however, was vague. The student used the word ‘security’ 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 better 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 IV) 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.

From the graphs, we analyzed how reflections influence the quality of design reasoning. We contrast the behavior of the test groups and the control groups. In the analysis, we also draw on the evidence from the explanatory effect of the transcripts. In summary, we have three findings, they are discussed in the following sections.

5.1. Finding 1—external reflection induces internal reflection

We observed that the test teams frequently make internal reflections, more so than the control teams. This is evident from Table V 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 induced the test teams to self-question during design.

Table VI 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 (Table VII) to illustrate how an external reflection induced internal reflections in a design session. Table VII 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 (Table VII) 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 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 solutionspace (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.

### 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 context and requirements loosely. Such poor design argumentation led to poor quality design discourse. We use examples from Transcript 3 (Table VIII) and Transcript 4 (Table IX) to illustrate how inaccurate context and requirements led to low quality design argumentation.

The students sought to ground the context and requirements using the admission business service as example (Transcript 3, Table IX 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 (Table IX) illustrates how inaccurate problem formulation leads to insufficient or irrelevant solutions.

<table>
<thead>
<tr>
<th>Design dialogue</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Lecturer</strong> For your business domain, what are the assumptions?</td>
<td>An external reflection of the context</td>
</tr>
<tr>
<td><strong>2. Student 1</strong> 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.</td>
<td>A description of what the system was about.</td>
</tr>
<tr>
<td><strong>3. Student 2</strong> 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?</td>
<td>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.</td>
</tr>
<tr>
<td><strong>4. Student 1</strong> Yes, maybe how do we ensure easy access to users with special needs?</td>
<td>Confirmation that the requirements need to be further explored.</td>
</tr>
<tr>
<td><strong>5. Student 3</strong> 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.</td>
<td>This student explored potential solution options.</td>
</tr>
<tr>
<td><strong>6. Student 1</strong> 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?</td>
<td>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.</td>
</tr>
<tr>
<td><strong>7. Student 3</strong> 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</td>
<td>This student applied the criteria to different groups of users to see if anything was amiss. This was an exploration of the system context.</td>
</tr>
</tbody>
</table>

Table VII. Transcript 2 interpretation: interaction of external and internal reflection in T2.

Table VIII. Transcript 3: inaccuracy in context and requirements in T1.

Table IX. Transcript 4: inaccuracy in problem formulation in T3.
Student 1 clarified the main purpose of their e-Service (Transcript 4, Table IX 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 option 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 analysis was lacking. If such was the case, we wanted to see how reflections or Mind 2 can help to improve the situation.

5.2.1. Finding 2.1—external reflection improves quality. Reflections bring to a design discourse, we need to capture the progression of quality in a design discourse, with and without reflections. The gray ZigZag curves in Figure 4 and Figure 5 show that the quality of design discourse fluctuates in both test team T1 and control team C3. When we examined the quality fluctuations during a design discourse, we noticed that there are delayed improvements after external or internal reflections rather than immediate improvements (see Figure 3). We evaluate the overall trends of quality progression of the design discourse visualizing quality progression trends using logarithmic regression analysis.

We plotted a trend curve for the quality progression of each team’s design discourse (for examples see red curves in Figure 4 and Figure 5). The resulting trend curves, shown in Table A3 and Table A4 in the Appendix, revealed the following interesting observation. There is a considerable improvement in the quality of the design discourse in the test teams. The red curve in Figure 4 shows a typical quality improvement when reflections are given, i.e. a progress in the quality. The only outliers were T3 and T7—not showing any quality improvements—for which we provide our explanation in Section 5.3. In contrast, the regression curves of control teams, shown in Table A4, revealed that the quality of the design discourse remains constant, over time. The red curve in Figure 5 shows a typical quality improvement when no reflections are given: no improvement.
We understand that the increase in the quality of design discourse in design teams could be
explained as simply designers 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 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 [78]. 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 [79].
Besides, we know from [18] that an ineffective design discourse can lead to an inferior outcome.

5.2.2. Finding 2.2—without internal reflection, design quality does not propagate. 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 A3). In contrast, when the number of internal reflections is small, we
observed that the accuracy improvements do not propagate. For example, in the graph shown in
Figure 6 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 propagate. 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.

5.3. Finding 3—reflection cannot improve quality without domain knowledge

The quality of design discourse is also dependent on appropriate knowledge designers possess.
Sometimes designers have knowledge about the domain but cannot recall the knowledge. By asking
questions like why is this an issue? or who cares about this problem? we prompted students to reflect on their context 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.

By analyzing the transcripts, we found some examples to show that the use of Mind 2 can help students to know whether they have the knowledge. Let us consider Transcript 5 (Table X) where students had some idea about the domain. The students were considering a system to keep track of the history of unemployed people. An external reflection was given to prompt why the students wanted to keep track of 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 context or the problems, then reflections help them to improve the relevance of a design discussion.

On the other hand, students that did not have relevant domain knowledge struggled to have a relevant design discussion. Consider Transcript 6 (XI), 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 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 in Table XI), then they cannot improve their design discourse. Therefore, only when the domain knowledge is sufficient would reflections improve the quality of a design discourse.

### 6. LIMITATIONS

Lincoln and Guba [77] 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.

#### 6.1. Objectivity

Objectivity ensures that the conclusions depend on the subjects and the conditions of the case rather than on the researcher [73]. 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 [73] 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.

6.2. Reliability

The underlying issue of reliability is whether the process of study is consistent over time across researchers and methods [73]. 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 4.4 the Cohen’s Kappa Coefficient [75] 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 [76]. 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.

Internal Validity Internal validity aims at ensuring that the collected data enables researchers to draw valid conclusions. We conducted multiple case studies 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 case studies. 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 practice sessions were 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 study. We mitigated this risk by random allocation of students to teams. There is however an inherent limitation to such random allocation as it does not ensure uniformity of group
behavior. This is because some confounding factors such as personality traits [80] or gender will inevitably have some effect in group’s dynamics [81] as well as group’s design discourse. For instance, whether group’s behavior is reflective or not will in part be influenced by the interactions within a group and who in the group are the dominant personalities.

The students were also told that their grading in the course was not depending on the performance in the practice session, only on their serious attendance. There is also the risk that the students lack motivation; they may consider their participation as non-rewarding. 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 event led to what, over each practice session.

6.3. 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 the 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, asking questions such as ‘What are the issues in the decision?’ and ‘What are the options to deal with the issues?’ have been shown to have an impact on design quality [6]. This work provides additional evidence to support the Two-Mind theory.

We tested our model with Masters students who, if not already working in industry, are close to finalizing their education. 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 [65] 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.

7. DISCUSSION

There are many definitions of reflection. In traditional software design methods, design review and walkthrough [14] share a similar purpose as reflection. An external reviewer can be used to review or walkthrough a design to check if anything is amiss. However, Mind 2 is a systematic process of exploring different aspects on design reasoning. As shown in Table I, we propose a theory to systematically explore different reasoning activities and reasoning techniques. Additionally, these mental activities are not done at a certain stage of design but throughout the design such that designers can be reflective during a design discourse. Therefore, we selected a narrow interpretation so that we can differentiate the act of reflection and the act of reasoning in our study. 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 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.

7.1. Reflection inertia

Our study suggests that the students, as novice designers, have a 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 [60]. 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. Satisficing behavior may also
have a role in such reasoning laziness. 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.

7.2. 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 [42] in that reflection questions and challenges design reasoning. The role of a reflection advocate is to ask questions similar to that in Table I to check the assumptions, risks, problem statements and other areas of decision making. In our work we acted in a similar manner to Scrum masters in the role as facilitators with the students to bring reflections and introspection explicit. However, we found that introducing reflections explicitly helped the students to reframe the problem and the solution architecture.

7.3. 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 a solution-driven strategy may not work very well. A software designer may need a scopingquestioning strategy in order to ‘get a better feel’ for what the problem is about [61]. Reflective design thinking is deliberate and it challenges the autonomous thinking one might easily take.

7.4. Reflection in real-world practice

The study of reflection in real-world environments is more difficult, as experience, self-belief and other factors can be difficult to isolate in the results. One research approach is the use of case study as it was done by Schön [45] 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 [15]. We intend to study experienced designers in a similar way but with reflections as a stimulus.

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 the designers; (ii) internal reflections were challenges self-raised by the designers. In this study, the test groups that had received reflections performed better than the control groups that had received no reflections.
APPENDIX A:
Diagrams

Table A1. Reflection and quality of design discourse in test teams.
Table A2. Reflection and quality of design discourse in control teams.
Table A3. Regression curves for test teams.
Table A4. Regression curves for control teams.
### Table A5. How do we handle data requests for external resources?

<table>
<thead>
<tr>
<th>Design issue</th>
<th>D6: How do we handle data requests for external resources?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Context</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Quality attributes</strong></td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td>Cr1: nfr 03 Consistency, Cr2: nfr 02 Performance</td>
</tr>
<tr>
<td>Description</td>
<td>D6-Opt1: sending request when needed.</td>
</tr>
<tr>
<td>Relationship(s)</td>
<td>—</td>
</tr>
<tr>
<td>Status</td>
<td>Rejected</td>
</tr>
<tr>
<td>Evaluation</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Architectural options</strong></td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td>D6-Opt2: Caching requested data.</td>
</tr>
<tr>
<td>Description</td>
<td>Data that has been requested within certain time interval is cached.</td>
</tr>
<tr>
<td>Relationships</td>
<td>D7 (What is the most appropriate time interval for caching different types of data?) D8 (What types of data can (and cannot) be cached)</td>
</tr>
<tr>
<td>Status</td>
<td>Accepted</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Cr1: This option could cause data inconsistence, because data might change while the system is in use. Cr2: This option reduces network traffic and consequently increases performance.</td>
</tr>
<tr>
<td>Rationale</td>
<td>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.</td>
</tr>
</tbody>
</table>
NOTES

1. Test teams are labeled with ‘T’ and control teams are labeled with ‘C’. For instance, we call test team 1 T1 and control team 1 C1.

2. It should be noted that the regression analysis was used to show trends and not for identifying the causality between reflection and quality.

REFERENCES


