Software Architecture Design Reasoning: A Card Game to Help Novice Designers

Courtney Schriek[1], Jan Martijn E.M. van der Werf[1], Antony Tang[2], and Floris Bex[1]

[1] Department of Information and Computing Science
Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
c.j.schriek@students.uu.nl, {j.m.e.m.vanderwerf, f.j.bex}@uu.nl
[2] Swinburne University of Technology, Melbourne, Australia
atang@swin.edu.au

Abstract. Software design is a complicated process, and novice designers have seldom been taught how to reason with a design. They use a naturalistic approach to work their way through software design. In order to impart the use of design techniques, a card game was developed to help design reasoning. This game was tested on groups of students and resulted in noticeable differences between the control and test groups. Those who used the cards produced better design arguments: the groups with the card game on average perform 75% more reasoning than the control groups. The results show that the design strategy used by the groups is a clear indicator for how many and what kind of design problems are designed, while the cards influence how the designers solve these problems

1 Introduction

Software architecture design is a complicated process, mostly revolving around problem-solving activities. Within software development many things have to be taken into consideration, not least being the requirements, but also what available technologies there are, the needs of stakeholders, and those of the developers and future redesign teams. This is known as a wicked problem, meaning that problem and solution are intertwined so that understanding the problem depends on how the designer wants to solve it. Such problems are inherently ill-defined and have no standard solution [17]. As a result it is the design decisions made at this stage that have the greatest influence on the eventual product. But people do not always use logical thinking, instead making decisions based on instinct, what is known as naturalistic decision making [10]. This can cause flawed reasoning, especially when the problem is complex and/or new, combined with the designers lack of expertise.

In order to resolve these problems, designers need to move from naturalistic decision making to logical reasoning decision making, especially when designers are not experienced with either the problem domain or the solution domain.

There has been some software design reasoning research, but there is no simple and comprehensive method that can be used in practice. In this paper, we propose a card game for this purpose. The card game combines techniques in a simple form, and is developed to help novice designers to consider certain reasoning steps during design.

This card game has been tested in an experiment involving students of a Software Architecture course. The aim of the experiment is to assess any obvious differences among control and test groups, and to establish if the card game had a positive influence on the logical reasoning process using qualitative analysis.

The card game is based on the reasoning techniques described in Tang and Lago [25]. The main reasoning techniques are the identification of assumptions, risks and constraints. The students also need to carry out trade-offs and articulate the contexts, problems, and solutions. Novice designers can forget that they need to reason about certain things. Razavian et al. [15] proposed to use a reflection system to remind designers. In their experiment, reflective thinking was applied using reasoning techniques to trigger student designers to use logical reasoning. In our experiment, cards were used to remind novice designers to use reasoning techniques. The choice for cards instead of a software tool was made because many software tools, such as those used for Design Rationale, have been developed but are not prevalently used in the design world. The most common reason for this is that the adoption and use of these systems take too much time and are too costly to be effectively used [24]. The cost-effectiveness of such a system isin fact the most important characteristic for consideration by software companies [12]. Cards do not cost much to produce or to use, and depending on the rules do not needextensive time to learn. Additionally, cards are not unfamiliar in software design, as already several card games exist and designers are familiar with their uses. For example, IDEO method cards are used for stimulating creativity [8], Smart Decisions card are used for learning about architecture design [3], and planning poker is played during release planning to estimate the time needed to implement a requirement [5]. Our card game additionally includes several reflection periods during the experiment to encourage participants toexplicitly reason with the design.

We compare novice designers equipped with reasoning techniques to the control groups using natural design thinking. The results show that the test groups came up with many more design ideas than the control group.

In the following sections more background information on design reasoning are given, together with reasoning techniques which stimulate logical reasoning (Sec. 2). Next, we introduce the student experiment, how it has been performed, and how validation of the results has been reached (Sec. 3). After this the results of the experiment will be explained, ending with an analysis of the results and further discussion on how the card game can be used for further experiments (Sec. 4). Threats to validity are discussed in Sec. 5. Section 6 concludes the paper.

2 Design Reasoning

Design reasoning depends on logical and rational thinking to support arguments and come to a decision. In a previous research, it has been found that many designers do not reason systematically and satisfice results easily [29]. Designers often use a naturalistic process when making decisions where experience and intuition play a much larger role [31]. As such, people need to be triggered to consider their decisions in a more rational manner. This can be done by using reasoning techniques. Previous research has shown that supporting designers with reasoning analysis techniques can positively influence design reasoning [15,27,28]. There are other issues with design thinking, which can be categorized as: cognitive bias, illogical reasoning, and low quality premises.

Cognitive bias occurs when judgments are distorted because the probability of something occurring is not inferred correctly or there is an intuitive bias. This can be seen with representativeness bias and availability bias, where the probability of an event is mistaken because it either looks more typical, or representative, or because it is more easily envisioned. An example is anchoring, where software designers choose solutions for a design problem based on familiarity, even when it is ill-suited to solve the problem [21,23].

Illogical reasoning is when the design reasoning process is not used and problems occur with identifying the relevant requirements. The basis premises and arguments being used in the design discussion are not based on facts.

Low quality premises for design argumentation can be caused by missing assumptions, constraints or context. Premises of poor quality can be caused by either an inaccurate personal belief or the premise being incomplete or missing. Much of reasoning depends on the quality of the premises themselves, if these are not explicitly stated or questioned, software designers are more likely to make incorrect decisions [23,26]. The basis of how such reasoning problems can develop lies in the difference between the two design thinking approaches: the naturalistic decision making, and the rational decision making. This is sometimes referred to as a dual thinking system: System 1 is fast and intuitive with unconscious processes, i.e., naturalistic decision making. System 2 is slow and deliberate with controlled processes, i.e., rational decision making [9]. People naturally defer to System 1 thinking, and so in the case of software design designers need to be triggered to use System 2 thinking for decision making. This is done by invoking reflective thinking or prompting, which in the simplest sense is thinking about what you are doing, meaning that the person consciously evaluates their ideas and decisions [18,22].

2.1 Design Reasoning Techniques

In order to trigger logical reasoning several design reasoning techniques can be implemented during design, such as risk analysis [14], trade-offs [1], assumption analysis [11], and problem structuring [16]. These reasoning techniques support logical reasoning by means of analysis of various aspects of design decisions. These techniques are well known. However, to combine these techniques in a simple form to teach and remind students to consider certain reasoning steps during design is new.

The reasoning techniques chosen for this experiment are not exhaustive and are instead a selection of common techniques already used in software architecture: problem structuring, option generation, constraint analysis, risk analysis, trade-off analysis and assumption analysis.

Problem structuring is the process of constructing the problem space by decomposing the design into smaller problems. These then lead to reasoning about requirements and the unknown aspects of the design [19]. This reasoning technique focuses on identifying design problems and how these can be resolved when the situation is one the designer is unfamiliar with. It is used to identify the problem space and the key issues in design by asking questions related to the problem, such as what are the key issues.

Its aim is to prevent the designer from overlooking key issues because of unfamiliarity with the problem. The more time spend on problem structuring the more rational an approach the designer uses.

Solution Option generation is a technique specifically directed at the problem of anchoring by designers, in which the first solution which comes to mind is implemented without considering other options. With option analysis the designer looks at each decision point at what options are available to solve a design problem.

Constraint analysis looks at the constraints exerted by the requirements, context and earlier design decisions and how they impact the design. These constraints are often tacit and should be explicitly expressed in order to take them into account. Trade-offs can come from conflicting constraints.

Risk analysis is a technique to identify any risks or unknowns which might adversely affect the design. Risks can come from the designer not being aware if the design would satisfy the requirements, in which case the design needs to be detailed in order to understand these risks and mitigate them. Or the design might not be implementable because designers are unaware of the business domain, technology being used and the skill set of the team. These risks should be explicated and estimated.

Trade-off analysis is a technique to help assess and make compromises when requirements or design issues conflict. It can be used for prioritization of problems and to weigh the pros and cons of a design which can be applied to all key decisions in a design [25].

Assumption analysis is a technique used to question the validity and accuracy of the premise of an argument or the requirements. It focusses mainly on finding hidden assumptions. It is a general technique which can be used in combination with the other reasoning techniques [23].

We propose a simple method that combines the main design reasoning techniques, and use a card game to prompt novice designers. In our research, we test the effectiveness of this technique.

3 Student Experiment

The theory studied in this paper is that applying reasoning techniques, through the use of a card game, has a positive influence on design reasoning with software designers. This theory is tested using an experiment focusing on inexperienced designers or novices. This experiment involved test and control groups carrying out a design. The results of the two groups are compared to one another. We use simple descriptive statistics and qualitative analysis to analyse the results.

The subjects for the experiment are 12 teams of Master students from the University of Utrecht, following a Software Architecture course. These were split into 6 control teams and 6 test teams, with most having three designers working together, two teams with two designers, and one team with four designers. Based on an earlier assessment, the teams were ranked, from which they were randomly selected for the test or control groups to ensure an equal amount of skill.

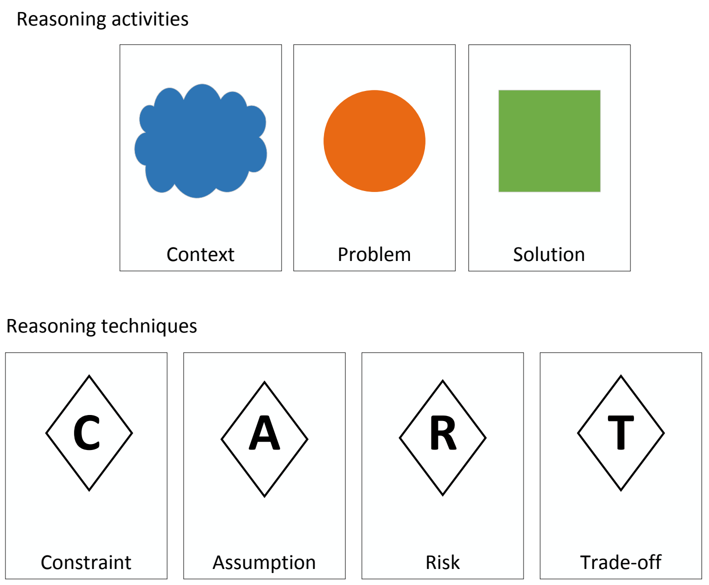

Fig. 1. Final cards of the card game

3.1 Experiment Design and Pilot Testing

Before the student experiment, a pilot study was run to test the card game and refine it by finding any major flaws or misunderstandings. The pilot study was performed by two Master students whom had already completed the course. The results of the pilot resulted in several important changes being made to the card game. Firstly, the cards were simplified and reduced to 7 cards, to simplify card play as the initial number of cards made it difficult to choose from them. The final card game is show in Fig. 1. The pilot showed that card play tapered off towards the end of the design session. To enforce card play, three reflective periods were added evenly spread throughout the session when cards have to be played. Lastly, the card rules were simplified to remove restrictions on fluid discussion. This resulted in the following set of playing rules:

1. The game is played in terms of a discussion;
2. The discussion starts when you play a card;
3. Others contribute to the discussion by playing their own cards;
4. When a decision is made the cards related to that topic can be removed.

The assignment used in the experiment is the same as used in the Irvine experiment performed at the University of California [30]. This assignment is well known in the field of design reasoning, as participants to the workshop analysed the transcripts made and submitted papers on the subject [13]. The assignment is to design a traffic simulator. Designers are provided with a problem description, requirements, and a description of the desired outcomes. The design session takes two hours. The assignment was slightly adjusted to include several viewpoints as end products in order to conform to the course material [1]. The sessions were recorded with audio only and transcribed by two researchers.

The card game is constructed based on an earlier experiment [15] which incorporated the reasoning techniques as reflective questions by an external observer who served as a reflection advocate to ask reflective questions. The card game replaces these questions with cards. Four of the reasoning techniques previously given were made directly into cards; *constraint*, *assumption*, *risk* and *trade-off*. Problem structuring and option generation would be triggered by using these techniques and looking at the design activities; context, problem and solution.

Three reflection periods were created at 15 mins, 45 mins and 1 hour and 45 mins. In these pre-set times, the students in the test groups were asked to use the cards to prompt discussion and support collaboration. The cards were paired with a table showing suggested questions to ask. Combining the cards enables different questions, such as: *which constraints cause design problems?* The control groups performed the same assignment without the card game, nor having pre-set reflection periods to revise their discussions.

A deductive analysis approach is used for coding the transcripts. The coding scheme is based on the design activities and reasoning techniques. The results of the experiment are analysed using qualitative measures, in this case with discourse analysis performed on the transcripts.

3.2 Results

In this section the results of the experiment are shown. The results show that there are significant differences between the control and test groups, supporting the theory that reasoning techniques influence design reasoning in a positive manner by having them use these techniques more.

Design Session Length. The first and most obvious difference is the time taken for the design session between the control and test groups. Though all groups were given two hours to complete their session, it is mostly the test group which took full advantage of this (Tbl. 1). Half of the control groups finished their design before the 1,5 hour mark. Only one test group did the same. From the audio recording, we conclude that this was due to a misunderstanding, as the group believed they had already breached the two hour mark. One test group even surpassed the two hour mark by almost half an hour.

Card Game Influence. To establish if there are any noticeable differences in the use of the reasoning techniques, the frequencies in which these were used are measured and compared (Tbl. 2). The results show that there is a noticeable difference in the frequencies in which the reasoning techniques are used. From the techniques directly influenced by the cards especially assumption and risk analysis are consistently more used by the test groups.

Table 1. Design session times

Control group	Time
C1	1:43:51
C2	1:57:15
C3	1:22:47
C4	1:13:39
C5	1:17:20
C6	2:05:33

Test group	Time
T1	2:01:23
T2	1:23:00
T3	1:59:56
T4	2:24:42
T5	1:54:48
T6	1:51:34

Table 2. Design reasoning techniques frequencies

Analysis techniques	T1	T2	T3	T4	T5	T6	Total
Assumption analysis	14	6	9	5	8	7	49
Constraint analysis	7	9	2	7	8	10	43
Risk analysis	6	7	6	5	5	5	34
Trade-off analysis	5	2	2	4	2	1	16
Option generation	19	2	11	6	6	8	52
Problem structuring	33	19	24	25	20	25	146
Total	84	45	54	52	49	56	340

Analysis techniques	C1	C2	C3	C4	C5	C6	Total
Assumption analysis	2	0	2	3	1	3	11
Constraint analysis	4	6	10	7	7	5	39
Risk analysis	2	2	3	4	1	3	15
Trade-off analysis	1	1	0	3	0	1	6
Option generation	1	3	4	6	9	7	30
Problem structuring	15	20	18	12	15	13	93
Total	25	39	37	35	33	32	194

Option generation and problem structuring, which are indirectly influenced by the cards, are also more prevalent with the test groups. The test groups on average perform 75% more reasoning than the control groups.

Constraint analysis on the other hand is about even among the test and control groups, and does not show the same rise in use of reasoning techniques. To better understand these results, we examine the distinct values of the individual reasoning elements. The reasoning techniques themselves are overarching and can contain several elements, and elements can be repeated. The distinct number of design elements identified by the two groups is given in Tbl. 3. The groups can repeatedly discuss the same assumption for instance, but the number of distinct assumptions identified shows that the reasoning techniques help designers find new design information. The distinct design elements are shown in the grey columns in Tbl. 3. The percentage difference shows 77% more identification of distinct reasoning elements by the test groups.

Our results show that assumptions and risks occur with a similar frequency as with their reasoning techniques. The constraints are shown to have an even more similar frequency across the test and control groups, there is hardly any difference at all. Although trade-off analysis shows an obvious difference, it is the lowest in frequency with both

Table 3. Design reasoning elements – No. of Distinct Identification

Design reasoning elements	T1		T2		T3		T4		T5		T6		Total distinct elements identified
Assumption	15	15	6	6	10	6	5	5	11	8	7	7	47
Constraint	9	8	17	8	5	4	9	7	13	8	18	7	42
Risk	6	6	7	6	7	5	7	7	7	7	8	6	37
Total													126

Design reasoning elements	C1		C2		C3		C4		C5		C6		Total distinct elements identified
Assumption	2	2	1	1	2	2	5	5	1	1	3	3	14
Constraint	4	4	12	8	17	9	9	5	22	8	16	8	42
Risk	2	2	2	2	3	3	4	4	1	1	3	3	15
Total													71

Table 4. Trade-off analysis, pros and cons elements

	T1	T2	T3	T4	T5	T6	Total
Tradeoff analysis	5	2	2	4	2	1	16
Pros	17	4	10	8	4	3	46
Cons	10	2	8	13	9	6	48

	C1	C2	C3	C4	C5	C6	Total
Tradeoff analysis	1	1	0	3	0	1	6
Pros	2	4	5	12	3	4	30
Cons	1	2	0	4	2	4	13

test and control groups. This is a surprising result as option generation shows a much greater difference in frequency. However, trade-off analysis, which concerns options, does not. To investigate these results we need to look at the elements which make up trade-offs; pros and cons (Tbl. 4). Taking a closer look towards the results, the differences between the test and control group becomes more obvious. The frequencies of pros and cons more closely match that of option generation. More pros and cons for various options are given; only the combination of both pro and con is scarce. As the coding scheme used requires a trade-off to have both a pro and a con for an option explains why trade-off analysis has such low frequencies. Interestingly, in comparison to the control groups, the test groups use both more pro with 53% more, but also far more cons to argue about their options, tripling the amount with 269% compared to the control group.

Looking at the identified design problems, options and solutions we find that mostly the design options have increased in the test groups compared to the control group, with a percentage difference of 56% (Tbl. 5). This corroborates with the increase in option generation established before. The identified design problems and solutions have increased with the test groups, but not by much: a percentage difference of 34% in design problems, and 24% with design solutions.

Table 5. Design problem, option and solution elements

	T1	T2	T3	T4	T5	T6	Total
Design Problems	29	10	17	17	8	13	94
Design Options	42	9	33	28	18	18	148
Design Solutions	29	10	17	17	8	11	92

	C1	C2	C3	C4	C5	C6	Total
Design Problems	3	8	13	19	16	11	70
Design Options	5	10	14	18	25	23	95
Design Solutions	4	9	13	20	17	11	74

4 Discussion

The results of the experiment show significant differences in applying reasoning between the control and test groups. The cards overall trigger more design reasoning in the test groups. More assumptions and risks are identified, more options are generated and more key issues are defined with problem structuring. In this section we analyse the results and discuss their meaning.

4.1 Thorough Reasoning vs Satisficing

A first result is the marked difference in the time spent in design. The test groups took longer for their design session, while the control groups took less time overall.

The test groups found more things to discuss and reason using the cards, whereas the control group is more partial to satisficing behaviour, which is a phenomenon where designers do not look exhaustively for every potential solution to a problem, but go with the first solution that is satisfying [20, 29]. This suggests that due to the cards, the test groups were reminded to reason with the reasoning topics, and were encouraged to explore more about the design. This supports our finding that **the card game leads to applying more reasoning techniques.**

The test groups were less easily satisfied with their decisions since they found more issues that they had to address. We can see this difference in attitude by examining the transcripts. The test groups often mention how they have run out of time before they are completely satisfied with their design. As can be seen in the extract of T5 (Fig. 2), a new design issue was mentioned, but there was no time to solve it.

The control groups on the other hand, especially those which did not reach the two hour mark, simply ran out of issues to resolve. In the extract of C5 (Fig. 2) they were touching on design issues that they needed to solve, but they convinced themselves that what they had was good enough (satisficing). They did not go further into detail to explore more about that decision but instead ended the discussion. Hence, **the card game combats satisficing behaviour.**

The control groups were easier satisfied with their decisions and design, even when they had not reached the full two hours given. For the test groups, the card game stimulated the designers to keep refining their design and consider their decisions, and often the time given was too short for these groups to fully explore the design.

4.2 How Cards Influence Design Discourse

The cards directly influence the design discourse in two ways. Firstly, the cards provide inspirations for students to investigate a certain topic. Secondly, the students use the cards to reassess their previous discussion by classifying it in card terms, e.g. a system rule is later identified as having been a constraint. Examples like the extract from T3 show how these cards are used for inspiration (Fig. 2). Person 2 was looking over the cards searching for issues to discuss and came up with a risk, which needed to be clarified for the other person. This risk made the designers reconsider an earlier assumption, that the program is a web-based application, which later turned into a nearly 5 minute long trade-off discussion.

With the extract from T2 we can see the cards being used for classification (Fig. 2). Here they had just discussed a problem and found a solution for it. But when they reassessed the discussion as a problem, they realized that in order to solve the problem, they had also identified risks and used assumptions.

4.3 Reasoning with Risk, Assumption and Trade-off

A main purpose of the reasoning card game is to prompt the students to consider design elements. The results of the experiment show that especially risks and assumptions are considered more by the test groups. Trade-off analysis does not show much difference, whereas constraint remained the same.

In many cases, the test groups considered the design scope to be clear at first glance. But when they started using the cards and thought more about the design topics, they found out that it actually is more complicated than they first realized. The designers reflect on their previous ideas, discuss and redefine them, which clearly shows that the cards trigger reasoning in designers.

For the control groups it is clear that considering assumptions and risks for decision making about the design is not at the forefront of their minds, as indicated by their low distinct element frequencies. With the test groups, the cards remind the designers to take these considerations into account, as again can be seen in T3 where person 2 lists the cards, which prompts him to identify a risk (Fig. 2).

For the trade-off analysis few pros and cons were discussed, contributing to the low number of trade-offs. However, the test groups generated many pros, and especially more cons to argue against the solution options than the control group. The control groups also generate many pros, but fewer cons (Tbl. 4). This suggests that the control groups are more concerned with arguments that support their options, or arguing why these are good, instead of looking at potential problems that could invalidate their options (cons). The test groups are more critical of their choices and look at options from different viewpoints. The extract of T4 shows part of a larger trade-off analysis in which several options are heavily discussed: mostly to have either a standalone program, or one which is cloud or web-based (Fig. 2). In this part, person 1 mentions that a pro for a cloud based program would be that you can update every hour, but a con is that a strong server is necessary which would be costly. The person then proceeds to suggest another option to ask users to pay for the usage of the server. This is not well-received by the

T5 (1:52:06-1:52:15)
PERSON 2: So we have we got everything. I think maybe only the traffic light is not taken into account and that's connected to intersection.
PERSON 1: Yeah. Definitely need to be there just make it here. And do we also model dependencies.
PERSON 2: Okay I think we don't have the time to put in. Maybe we can sketch it.

C5 (1:16:38 1:17:19)
PERSON 2: Oh ok. Do we have to say something more? Are we done actually? Or do they actually also wanna know how we include the notation and such, because-
PERSON 1: No they also get the documents, so they can see
PERSON 2: Yeah ok, but maybe how we come up with the- I dont know. No? isnt necessary?
PERSON 3: Mm
PERSON 1: Its just use UML notation, for all
PERSON 2: For all?
PERSON 1: No, and lifecycle model, and petri net. No, no petri net
PERSON 2: Perhaps petri net. Ok, shall we- shall I just?
PERSON 1: Yeah
PERSON 2: Ok

T3 (0:20:31-0:21:10)
PERSON 2: HTML 5 yeah? Information would of course [inaudible] constraints or risk or trade-offs, we have to make- a risk might be of course that- of course there is a [inaudible] so while you are travelling. For example, when you have an older device that could be a problem of course. So then you couldnt use the navigation maybe, the- well, [inaudible] right?
PERSON 1: What do you mean exactly? For example.
PERSON 2: Yeah well, for example, if you are travelling and you want to use the
application. You want to use the traffic simulator, then of course that might be the case that your device is not suitable for it. For example. So, on the other hand

T2 (0:28:14-0:28:28)
PERSON 1: So this was a problem
PERSON 3: This was a problem
PERSON 1: Yeah
PERSON 2: Yeah. Because [inaudible]
PERSON 1: And a risk right
PERSON 2: A constraint? Yeah but it was also like an assumption that you have a minimum length. That is our assumption right or-
PERSON 3: Yeah we created that now, and thats ok because its our own system

T4 (1:25:13-1:26:05)
PERSON 1: So that's the trade-off. The other side is good to have in the cloud because you can easily push a new update every hour if you want but you need really really strong server for all this simulations. Now professor did not say how much money she has. So it can be also. There can be also an option to pay for usage of this server for every simulation or for every hour of simulation.
PERSON 2: I don't think so.
PERSON 1: There can be an option. But it can be also very expensive so when I think about everything I think that is cheaper and easier to have local stand-alone version.
PERSON 2: Yeah.
PERSON 3: Yeah.

Fig. 2. Transcript Extracts

group and person 1 admits that this option would still be a very expensive one and gives a pro to their first option: a local standalone version to which the others agree.

Even though the group eventually went with their first option, they took the time to explore multiple options and critically assess them by providing both pros and cons. The control groups had fewer of such discussions.

4.4 Reasoning with Design Context, Problems and Solutions

The effect of the card game is to combat satisficing behaviour and lack of design reasoning by stimulating the designers to reconsider their options and decisions, ultimately taking more time to delve into the issues. And yet, when we look at the design problems and design solutions identified by both groups, the percentage difference is much lower than that of the other elements, such as options and problems structuring. The cards prompt designers to consider their problems and explore more of the design, but problems are not identified as much by the test groups, as the other reasoning techniques are used.

Design problems identification can be influenced by other factors, such as design strategy and designer experience. Design strategies such as problem-oriented, or solution-oriented can influence the information seeking behaviour of designers [16]. The approach used for problem-solving, whether to focus on finding problems or solutions first, seems to have more of an impact on the design problems being identified. When comparing groups with similar strategies, the influence of the cards becomes clearer.

As an example, we have groups T2 and C1. Both use a satisficing strategy, where they actively avoided going into the details. They preferred to view a problem as being outside of their scope. Their option generation and trade-off analysis results are very similar. But the problem structuring, risk, assumption and constraint analysis of T2 is at least double of that of C1. Despite their adherence to a minimum satisficing strategy, the cards prompted T2 to recognize problems which often resulted from identified risks and constraints, for which they made assumptions to simplify the problem and solution.

It seems that the design strategy used by the groups is a clearer indicator for how many and what kind of design problems are identified, while the cards influence how the designers solve these problems. This supports our finding that **problem identification depends more on the design strategy than on the card game.**

4.5 Constraint Identification

The card game seems to have no influence on constraint analysis. The individual constraints identified by both groups are the same. This result in itself is interesting, considering the effect of the cards on the other reasoning techniques. The question here is why constraint analysis is different. One possible explanation for this is that the very nature of constraints, i.e. limitations on the design, as seen by novice designers, is intrinsically bound to the requirements. When thinking about design and what the system must accomplish, novice designers think of what is required, and what is not required. As a result both test and control groups identify constraints as things that are not allowed or rules that the system must follow. What is interesting here is that both groups identify much of the same constraints, with many coming directly from the requirements in the assignment, even taking on the same wording.

We find that both test and control groups frequently take over the literal requirements presented in the text as constraints. To give a more detailed representation of this, for the test groups there are 11 identified constraints which are shared in various degrees amongst the groups. There are 11 other constraints which they do not share and had to be inferred from the assignment, with 5 of these being identified by only one group. The control groups share 12 constraints from the text, and only 5 are other.

This then goes to explain the similar results when it comes to constraint analysis, many of them are found literally in the text of the assignment and require minimal effort to find. It is easy to see why these constraints would be in the text as requirements, as it is to the clients benefit to give clear instructions on what the program should and should not do. This means that constraints are easier to identify, causing the cards to have little influence, as these are given as requirements. **The card game provides no noticeable difference in constraint identification.** The other techniques, such as assumptions and risks, must all be inferred from the text and are not clearly given. The effect of the cards is more obviously shown there.

5 Threats to Validity

We recognise the threats to validity in this research, especially those revolving around generalization. For the transcripts, discourse analysis was used to interpret the text, which in itself is subjective and reliant on the view of the researcher [7]. This paper is an empirical research in the form of an experiment involving an example assignment. Empirical research is one of the main research methods within the software architecture research field, relying on evidence to support the research results. We address the internal and external validity of the results acknowledging any limitations which may apply [4].

5.1 Internal Validity

Internal validity is about how far a valid conclusion can be made from the experiment and data collected. Firstly, this research makes use of the Irvine assignment which has been used and tested in other research and is well-known in the field of design reasoning [13]. This limits the results of this research to those applicable to this kind of design assignment.

Secondly, participants were randomly selected for their situational representativeness, as students of software architecture, and the result of this research is limited to novice designers in the Netherlands. However, we have found convincing results to show that the card game made a difference to the reasoning capability of novice designers. We argue that these two limitations do not impose a major threat to the interpretation of the evidence that the cards have a positive effect on design reasoning by novices.

5.2 External validity

External validity is about to what extent the results from the case study can be generalized across other published design reasoning articles. The results of this case study are supported by similar experiments [6, 15, 27, 28], showing that in the case of novice designers, being made aware of reasoning techniques actively counteracts satisficing behaviour and results in performing more design reasoning.

But the results also show a discrepancy when it comes to constraints, which did not show any difference across test and control groups. There are two possibilities for this discrepancy, either a requirement naturally leads to constraints, or the assignment itself is too clearly defined by explicitly including constraints. Whether the constraint card would have any influence on an assignment which did not mention constraints in their requirements cannot be proven at this point.

For the design problems and solutions there seems to be a design strategy component which has an influence on the amount of design problems being identified. This makes it unclear how much of an influence the cards have.

5.3 Reliability

Reliability is about ensuring that the results found in the study are consistent, and would be the same if the study is conducted again. To ensure that the coding of the transcripts is reliable it was tested for inter-reliability using Cohens kappa coefficient [2] to measure the level of agreement. The transcripts were each coded by two researchers using Nvivo 10. The average kappa coefficient of each of the transcripts was above 0.6 which is considered to show a good level of agreement. The average of all transcripts combined is 0.64.

6 Conclusions

Software design is a complicated problem-solving process, which due to its effect on the later stages of software development, is one of the most important stages to consider. Problems occurring at this stage which are not solved immediately will result in problems later during development or implementation, costing money and time. Problems with software design can result from problematic design decisions, which are easily influenced by designer biases. These biases can be avoided by using more logical reasoning.

In this paper, we propose a simple card game to help novice designers use design reasoning. Design reasoning means using logic and rational thinking in order to make decisions, something which people as a whole find difficult due to the usual way they think. In order to prompt design reasoning several common reasoning techniques were chosen to be represented by the card game. These techniques are; problem structuring, option generation, constraint analysis, risk analysis, trade-off analysis, and assumption analysis.

To study the effect of the card game, we designed an experiment based on 12 student groups following a software architecture course. These 12 groups were divided into 6 control and 6 test groups. The 12 groups were asked to construct a software design. The transcripts of these experiments were analysed using discourse analysis. The results show a notable difference between the test and control groups on nearly all technique usages. The effect of the cards is to trigger the designers to use design reasoning techniques to reason with different aspects of design, to prompt new discussion topics, or to reconsider previous discussions. In all manners, the cards trigger reasoning and lead to more discussion and reconsideration of previous decisions. Those who use the card game generally identify more distinct design elements and spend more time reasoning with the design. Only the constraint analysis technique shows no obvious difference.

Further research includes to study the effect of the card game to professional designers, i.e., those who are experienced in the field. Professionals have more experience. Therefore, it would be interesting to observe how such a simple card game works with people who are more aware of design techniques. The card game could also be used as a learning tool for novice designers, to further their understanding of software architecture and learn design issues from the reasoning angles.

References

1. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Series in Software Engineering. Addison Wesley, Reading, MA, USA, 2012.
2. J. Cohen. Weighted kappa: nominal scale agreement with provision for scaled disagreement or partial credit. *Psychol. Bull.*, 70:213–220, 1968.
3. Smart Decisions. A software architecture design game. http://smartdecisionsgame.com/.
4. M. Galster and D. Weyns. Emperical research in software architecture. In *13th Working IEEE/IFIP Conference on Software Architecture, Italy, Venice.*, pages 11–20. IEEE Computer Society, 2016.
5. J. Grenning. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods Renaiss. Softw. Consult.*, 1-3, 2002.
6. U. van Heesch, P. Avgeriou, and A. Tang. Does decision documentation help junior designers rationalize their decisions? a comparative multiple-case study. *J. Syst. Softw.*, 86:1545–1565, 2013.
7. D. Horsburgh. Evaluation of qualitative research. *J. Clin. Nurs.*, 12:307–312, 2003.
8. IDEO. IDEO Method Cards. https://www.ideo.com/by-ideo/method-cards/.
9. D. Kahneman. *Thinking, Fast and Slow*. Penguin Books, UK, London, 2012.
10. G. Klein. Naturalistic decision making. *Hum. Factors J. Hum. Factors Ergon. Soc.*, 50:456–460, 2008.
11. P. Lago and H. van Vliet. Explicit assumptions enrich architectural models. In *27th International Conference on Software Engineering, 2005. ICSE 05*, pages 206–214. ACM, 2005.
12. J. Lee. Design rationale systems: Understanding the issues. *IEEE Expert. Syst. their Appl.*, 12:78–85, 1997.
13. M. Petre and A. van der Hoek. *Software Designers in Action: A Human-Centric Look at Design Work*. CRC Press, 2013.
14. E. R. Poort and H. van Vliet. Architecting as a risk- and cost management discipline. In *9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011*, pages 2–11. IEEE Computer Society, 2011.
15. M. Razavian, A. Tang, R. Capilla, and P. Lago. In two minds: How reflections influence software design thinking. *J. Softw. Evol. Process*, 6:394–426, 2016.

16. J. Restrepo and H. Christiaans. Problem structuring and information access in design. *J. Des. Res.*, 4:1551–1569, 2004.
17. H. W. J. Rittel and M. M. Webber. Dilemnas in a general theory of planning. *Policy Sci.*, 4:155–168, 1973.
18. D. A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. BasicBooks, USA, New York, 1983.
19. H. A. Simon. The structure of ill structured problems. *Artif. Intell.*, 4:181–201, 1973.
20. H. A. Simon. Rationality as process and as product of a thought. *Am. Econ. Rev.*, 68:1–16, 1978.
21. W. Stacy and J. MacMillan. Cognitive bias in software engineering. *Commun. ACM.*, 38:57–63, 1995.
22. K.E. Stanovich. Distinguishing the reflective, algorithmic, and autonomous minds: Is it time for a tri-process theory? In *In Two Minds: Dual Processes and Beyond*, pages 55–88. Oxford University Press, 2009.
23. A. Tang. Software designers, are you biased? In *6th International Workshop on SHAring and Reusing Architectural Knowledge*, pages 1–8. ACM, USA, New York, 2011.
24. A. Tang, M. A. Babar, I. Gorton, and J. Han. A survey of architecture design rationale. *J. Syst. Softw.*, 79:1792–1804, 2006.
25. A. Tang and P. Lago. Notes on design reasoning techniques, 2010. Swimburne University of Technology, SUTICT-TR2010.01.
26. A. Tang and M. F. Lau. Software architecture review by association. *J. Syst. Softw.*, 88:87–101, 2014.
27. A. Tang, M.H. Tran, J. Han, and H. van Vliet. Design reasoning improves software design quality. In *Quality of Software Architectures*, volume 5581 of *LNCS*, pages 28–42. Springer, Berlin, 2008.
28. A. Tang and H. van Vliet. Software architecture design reasoning. *Software Architecture Knowledge Management*, pages 155–174, 2009.
29. A. Tang and H. van Vliet. Software designers satisfice. In *Software Architecture: 9th European Conference, ECSA 2015*, volume 9278 of *LNCS*, pages 105–120. Springer, Berlin, 2015.
30. UCI. Studying professional software design. http://www.ics.uci.edu/design-workshop/.
31. C. Zannier, M. Chiasson, and F. Maurer. A model of design decision making based on empirical results of interviews with software designers. *Inf. Softw. Technol.*, 49:637–653, 2007.