International Journal of Software Engineering
and Knowledge Engineering
Vol. 24, No. 4 (2014) 553–589
© World Scientific Publishing Company
DOI: 10.1142/S0218194014500211

World Scientific
www.worldscientific.com

Validating and Improving a Knowledge Acquisition Approach for Architectural Decisions

Dan Tofan[*] and Paris Avgeriou[†]
Department of Mathematics and Computing Science
University of Groningen
Groningen, The Netherlands
[*]*d.c.tofan@rug.nl*
[†]*paris@cs.rug.nl*

Matthias Galster
Department of Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
matthias.galster@canterbury.ac.nz

Received 12 September 2013
Revised 17 November 2013
Accepted 13 February 2014

Software architects make architectural decisions such as choosing architecture patterns or frameworks. Capturing architectural decisions reduces evolution costs for software systems. Existing approaches overlook the challenge of capturing tacit knowledge about architectural decisions. Previously, we proposed the REGAIN approach to capture tacit knowledge about architectural decisions. REGAIN is based on the Repertory Grid technique, a powerful knowledge acquisition approach from knowledge engineering. However, REGAIN lacked industrial validation. Therefore, in this paper, we initially present a study to ensure that REGAIN meets the needs of industrial architects. We interviewed 16 architects who indicated REGAIN advantages such as systematic decision-making support. Also, architects indicated improvement opportunities, in particular tool support and the possibility to prioritize concerns that are used to evaluate decision alternatives. Therefore, we conducted an additional study to evaluate two approaches for prioritizing concerns: pairwise-comparisons and the hundred-dollar approach. We conducted an experiment with 30 graduate students to compare the two prioritization approaches. Based on the results of the experiment, we added the hundred-dollar approach to REGAIN. Moreover, based on the study with architects and the experiment with students, we implemented open source tool support for REGAIN, which architects can use to capture architectural decisions.

Keywords: Software architecture; decisions; documentation; interview study; experiment; prioritization; tool support.

1. Introduction

Software architects make decisions that influence the fundamental structure and behavior of software systems. For example, architects select programming languages, database systems or decompose systems into modules. For these decisions, architects need to balance stakeholders' concerns, such as quality attributes, business goals and functional requirements. Architects negotiate alternatives with stakeholders, considering tradeoffs among alternatives, based on how well alternatives satisfy concerns. Such tradeoffs are part of the rationale of architectural decisions.

Architectural decisions are the most important type of architectural knowledge. Information about architectural decisions and their rationale is often lost over time, leading to architectural knowledge vaporization [1, 2]. Knowledge vaporization increases evolution costs of a software system, because modifying previously made decisions becomes increasingly costly to assess and to implement. For example, implementing new features in a software system requires understanding the previously made architectural decisions. However, if the documentation is missing, incomplete, or outdated, then architects and developers need to spend substantial effort to understand previous architectural decisions, to avoid introducing conflicts with them.

In order to reduce knowledge vaporization, researchers proposed various approaches to capture architectural knowledge. Since architectural decisions are the most important part of architectural knowledge, most efforts focused on capturing architectural decisions. For example, Kruchten *et al.* describe a decision view embedded in the 4+1 view model [3]. Tyree and Akerman propose a template for describing decisions [4]. Van Heesch *et al.* propose four viewpoints for documenting architecture decisions [5]. Capilla *et al.* describe a software tool for capturing architectural decisions [6].

Existing approaches overlook some of the inherent challenges of capturing knowledge, which are widely recognized in other disciplines [7]. In particular, articulating tacit knowledge is very difficult [7], and existing approaches in software architecture overlook capturing tacit architectural knowledge [8]. Overlooking this challenge limits the practical usefulness of approaches for capturing architectural knowledge. To address the challenge of capturing tacit architectural knowledge, we argue for using ideas from the knowledge engineering discipline, because this discipline focuses on knowledge-related approaches, so it may offer ideas with high potential for tackling the problem of capturing tacit architectural knowledge.

One such idea is to use proven knowledge acquisition approaches for capturing tacit architectural knowledge. The Repertory Grid technique is such an approach: it is a powerful interviewing technique for tacit knowledge acquisition [9, 10]. In our previous work, we proposed an initial version of REGAIN (REpertory Grid for capturing ArchItectural decisioNs, which is presented in detail in Sec. 2), an approach which utilizes the Repertory Grid technique for capturing architectural decisions [11, 12].

However, REGAIN lacked industrial validation. Therefore, this follow-up paper investigates and improves REGAIN in industrial practice, to ensure that REGAIN meets the needs of industrial architects for capturing tacit architectural knowledge. In detail, this paper makes two main contributions: (1) it reports feedback from practitioners on using REGAIN for capturing real-world architectural decisions, (2) it presents an improvement of REGAIN by adding a prioritization approach that has been selected based on empirical data from an experiment. An additional contribution of this paper is open-source tool support for REGAIN. These contributions help address the challenge of capturing tacit architectural knowledge in industrial practice, and encourage using ideas from the knowledge engineering discipline to tackle the problem of architectural knowledge vaporization.

Figure 1 shows the three research phases that we followed, and the corresponding sections of this paper where the phases are presented. Phase 1 (in Sec. 2) presents the initial REGAIN approach as published in our previous work, background information on the Repertory Grid technique, and a summary of two previous preliminary evaluations of the initial REGAIN approach.

Phase 2 (in Sec. 3) covers REGAIN's industrial applicability from conducting an interview study with practitioners, in which they used the initial REGAIN approach, and offered feedback about it. In Phase 3 (Sec. 4), we improve the REGAIN approach by addressing two issues raised in Phase 2: the need for prioritizing concerns and tool support for REGAIN. We discuss validity threats in Sec. 5 and related work in Sec. 6. Conclusion and future work are presented in Sec. 7.

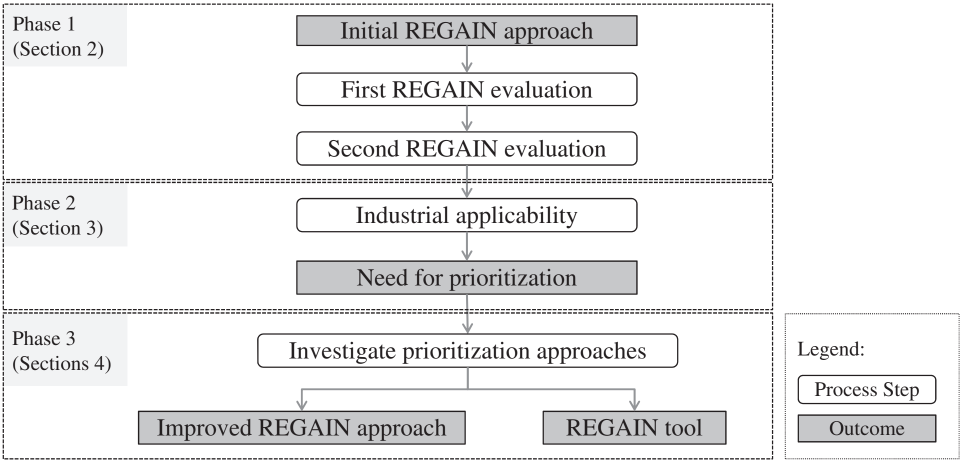


Fig. 1. Paper structure: Phase 1 was reported in [11, 12], Phases 2 and 3 are reported in this paper.

2. Phase 1 — Initial REGAIN Approach

In this section, we present REGAIN and our previous evaluations of REGAIN.

2.1. *Theoretical foundations for REGAIN*

REGAIN is based on two major theoretical foundations: the core model of architectural knowledge in [13], and the Repertory Grid Technique.

The core model of architectural knowledge in [13] specifies a minimalistic set of concepts for the architectural knowledge domain. The model aims at improving architectural knowledge management practices in the industry, such as capturing architectural knowledge [13]. Similar to other work [14, 15], the core model indicates that architectural decisions are a critical component of architectural knowledge.

Figure 2 shows the most relevant concepts for REGAIN from the core model in [13]. Compared to the original core model, we renamed the *rank* action to the *evaluate* action, and the *ranking* element with the *rating* element. When making an architectural decision, there is a topic, n alternatives, m concerns, and $n \times m$ ratings of alternatives against concerns. For example, a decision on the topic of choosing a database can include three databases (such as MySQL, MS SQL, and PostgreSQL), five concerns (such as scalability, availability of documentation, performance, familiarity, and costs), and fifteen ratings for each database against each concern (e.g. using a five-point scale). The actual decision is the alternative that satisfies best the concerns. The rationale for the decision is captured in the evaluations of alternatives against concerns [13] (i.e. the ratings indicate which alternative satisfies best the concerns).

The concepts in the core model in [13] have equivalent concepts in other established models for architectural knowledge: the IEEE 1471-2000 standard [16], Tyree's template [4], and Kruchten's ontology [17]. The equivalency among concepts is detailed in [13], and we offer below the following three examples.

(1) The concept of *concern* in the core model [13] is equivalent to the concepts of *requirement* and *risk* in Kruchten's ontology [17], and to the concepts of *assumption* and *constraint* in Tyree's template [4].

(2) Since architecting is an iterative activity, a *decision loop* appears: starting from an initial decision topic, the architect makes an architectural decision, which, in

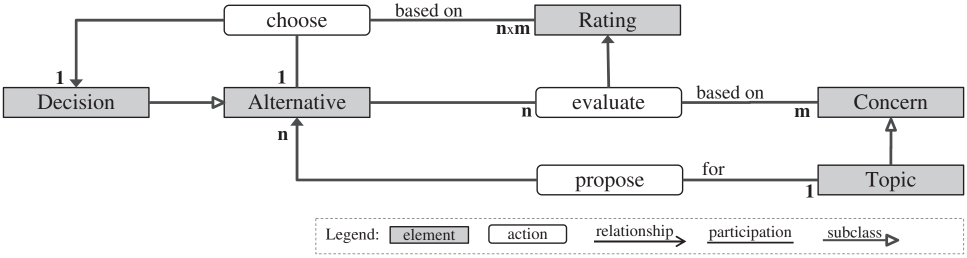


Fig. 2. REGAIN uses these concepts from the core model in [13].

turn, may uncover new concerns and decision topics, for which subsequent related decisions need to be made. The *decision loop* is equivalent to the *relationships* among decisions in Kruchten's ontology [17]. For example, the '*must use J2EE*' decision introduces a constraint relationship with the '*use the GlassFish application server*' decision in Kruchten's ontology [17], while in the core model [13] the '*must use J2EE*' decision introduces the '*which application server to use?*' decision topic, for which *GlassFish* is one of the considered alternatives. Overall, the *decision loop* enables traceability among decisions.

(3) The decision *rationale* concept in the IEEE 1471-2000 standard [16] is equivalent to the *evaluations* of alternatives against concerns in the core model in [13]. The *evaluations* explain why the architect selected a certain alternative.

In addition to the core model in [13], the other major theoretical foundation of REGAIN is the Repertory Grid technique. This technique is a structured interviewing technique that helps '*describe how people think about the phenomena in their world*' [18], while reducing the influence of the interviewer's own viewpoint [19]. The Repertory Grid technique helps experts make their tacit knowledge explicit [7], by offering a process centered around asking experts to compare repeatedly relevant items (e.g. decision alternatives) and indicate the concerns that differentiate the items. Since the technique is structured, it is suitable for automation by providing tool support to facilitate using the technique, and to reduce the need for an interviewer [19]. The process has proved easier for experts and more accurate compared to indicating directly (i.e. without any help) items and concerns [10].

2.2. *The REGAIN approach*

The REGAIN approach consists of the following five steps.

In **step 1** of REGAIN, the architect identifies the architectural decision topic (e.g. choice of a framework for developing a new web application).

In **step 2**, the architect generates a list of alternatives, relevant for the decision topic (e.g. a list of web frameworks). Alternatives may be reused from previous similar decision. Optionally, a hypothetical ideal alternative may be included (e.g. ideal web framework). The ideal alternative would represent the most preferred ratings for each concern in an 'ideal' world, without any tradeoffs. Comparing alternatives against the ideal alternative provides a basic utility function: alternatives with ratings more similar to the ideal alternative offer more utility than alternatives with less similar ratings.

In **step 3**, the architect provides concerns that are used to evaluate alternatives identified in step 2. The architect can reuse concerns from previous decisions (e.g. through keywords-based suggestions offered by tool support). Otherwise, the architect provides concerns using the triadic elicitation. Triadic elicitation means that the architect selects three random alternatives and then asks how two alternatives are similar, but different from the third. For example, for the decision topic on a Python web framework, an architect might consider Web2py and Zope similar in terms of popularity (i.e. both are used by *fewer popular existing websites*) and Django different with regard to popularity (i.e. there are *more popular existing websites* implemented with Django). Thus, the architect captures *popularity* as a concern in terms of a contrast (*fewer* and *more popular existing websites*). Repeated comparisons among other triads of alternatives produce more contrasts to express the concerns.

Triadic elicitations of concerns is the core step for making tacit knowledge explicit in a systematic manner. Jankowicz [20] indicates three reasons for this. First, triadic elicitations encourage expressing concerns as concrete contrasts (or poles) between the alternatives, instead of only labels for concerns (e.g. *popularity*). Second, triadic elicitations encourage describing contrasts in a precise and operational manner. For example, an architect can interpret popularity in terms of how many search results are returned for the names of the frameworks, so the architect should express his definition of popularity as a contrast and indicate as precisely as possible the values for the left and right poles of the contrast. Third, triadic elicitations encourage making explicit subjective concerns, such as architects' intuitions or gut feelings about the alternatives.

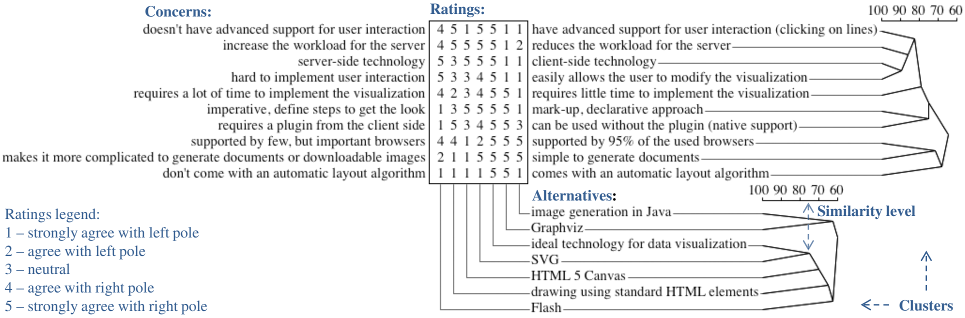
In **step 4**, the architect judges each alternative against every concern and captures the judgment in a rating. A typical rating scale uses integers from one to five, to help the architect indicate how an alternative satisfies a concern (e.g. assigning one for an alternative being *less popular*, or assigning five for being *very popular*).

In **step 5**, the architect analyzes the resulting grid, using *content* and *structure analysis* [19]. *Content analysis* refers to evaluating the decision topic (e.g. is this topic relevant), alternatives (e.g. should other alternatives be added to the grid), concerns (e.g. can some concerns be added or removed), and ratings (e.g. are there missing ratings, or do ratings need to be changed). The content analysis helps the architect understand if all components (e.g. alternatives, concerns) of the decision were included in the grid. *Structure analysis* of a grid uses cluster [21] and principal components [19] analyses. Both involve statistical operations for analyzing relationships between decision alternatives and concerns. This helps the architect make the decision. The result of cluster analysis is part of the output of REGAIN.

2.3. *REGAIN output*

The output of REGAIN consists of two main artifacts. First, the grid (or matrix) contains alternatives, concerns, and ratings. Second, two dendrograms (i.e. trees with relationship similarities between items) are produced from the cluster analysis of alternatives and concerns. The cluster analysis uses a nearest neighbor hierarchical clustering algorithm, with the city-block distance [21].

Figure 3 presents the grid and dendrograms (i.e. clusters) for an architectural decision collected in our previous work [11]. The decision topic was technology for data visualization. The REGAIN session produced seven alternatives (lower part of Fig. 3) and ten concerns (upper part of Fig. 3). As part of the analysis, alternatives and concerns are reordered and grouped based on their similarity. The level of



Concerns:	Ratings:							
doesn't have advanced support for user interaction	4	5	1	5	5	1	1	have advanced support for user interaction (clicking on lines)
increase the workload for the server	4	5	5	5	5	1	2	reduces the workload for the server
server-side technology	5	3	5	5	5	1	1	client-side technology
hard to implement user interaction	5	3	3	4	5	1	1	easily allows the user to modify the visualization
requires a lot of time to implement the visualization	4	2	3	4	5	5	1	requires little time to implement the visualization
imperative, define steps to get the look	1	3	5	5	5	5	1	mark-up, declarative approach
requires a plugin from the client side	1	5	3	4	5	5	3	can be used without the plugin (native support)
supported by few, but important browsers	4	4	1	2	5	5	5	supported by 95% of the used browsers
makes it more complicated to generate documents or downloadable images	2	1	1	5	5	5	5	simple to generate documents
don't come with an automatic layout algorithm	1	1	1	1	5	5	1	comes with an automatic layout algorithm

Alternatives:
image generation in Java
Graphviz
ideal technology for data visualization
SVG
HTML 5 Canvas
drawing using standard HTML elements
Flash

Ratings legend:
1 – strongly agree with left pole
2 – agree with left pole
3 – neutral
4 – agree with right pole
5 – strongly agree with right pole

Fig. 3. Grid of an architectural decision and the clusters (or dendrograms) for alternatives and concerns.

similarity is calculated using the distances between ratings. Alternatives and concerns are grouped in clusters, based on their similarity levels. A similarity level of 100% between two alternatives means that the two alternatives have the same ratings for all concerns. Based on the city-block distance, in Fig. 3, SVG (i.e. Scalable Vector Graphics) is the most similar alternative to the ideal technology (around 75%, which is the highest similarity level between any pair of alternatives). For the architectural decision in this grid, the practitioner chose the SVG alternative, which is the closest alternative to the ideal technology.

2.4. *REGAIN evaluations*

We reported two previous studies with evaluations of the initial REGAIN approach in [11, 12], as shown in Fig. 1. The first study [11] aimed at exploring the feasibility of the initial REGAIN approach. To this end, we conducted an exploratory study with seven graduate and undergraduate students, who used REGAIN to capture architectural decisions from their previous academic and industrial projects. After students had captured decisions, we interviewed them to obtain feedback on advantages and disadvantages of REGAIN. We found out that the approach helps students make architectural decisions in a systematic manner, and that it encourages students to reflect on their decisions. Also, the approach produces a concise documentation of architectural decisions, which reduces architectural knowledge vaporization.

In the second study [12] we investigated if the REGAIN approach reduces architectural knowledge vaporization. In the study, we asked twenty graduate students to capture architectural decisions with the initial REGAIN approach (similar as in the first study). Next, we compared the output of capturing decisions with REGAIN with the architectural documentation on the same decisions that students created as part of a software architecture course project. For the course project, students received a template for documenting the architecture of a software system. The template included sections for documenting architectural decisions, to indicate decision topic and rationale. Although students spent much time on the course project architectural documentation, we found out that REGAIN was more efficient (i.e. it took less time), and more effective (i.e. it captured more architectural knowledge, in terms of alternatives, concerns, and evaluations of alternatives against concerns), compared to the documentation with a template for documenting architectural decisions. The explanation is that REGAIN is based on the minimalist core model of architectural knowledge in [13], and REGAIN offers a structured approach with clear steps, which encourages architects to capture the most relevant architectural knowledge (e.g. alternatives, concerns). Overall, the study results confirmed that REGAIN reduces architectural knowledge vaporization in an efficient manner.

The results from the two evaluations with students [11, 12] laid the groundwork to proceed to the next research phase: investigating the industrial applicability of REGAIN.

3. Phase 2 — Investigate Industrial Applicability of REGAIN

3.1. *Research method, data collection and analysis*

The research goal of Phase 2 was to obtain feedback on REGAIN from industry, for the purpose of understanding advantages, disadvantages, and potential improvements to REGAIN, from the viewpoint of practitioners (i.e. architects in the industry), in the context of architects' efforts for capturing architectural decisions. To achieve the stated research goal, we formulated the following research questions.

RQ1. What are the advantages and disadvantages of REGAIN?
RQ2. What are the improvement opportunities for REGAIN?

To answer these research questions, we conducted an interview study with architects from the industry. We selected architects with a wide range of experience levels, which is representative for the situation in the industry, where architects have various experience levels. We contacted architects in our network and architects from open source projects (e.g. SOFA Statistics project at www.sofastatistics.com). Upon a positive response, we conducted individual interview sessions with them, to capture their architectural decisions with REGAIN.

Each session took typically two hours, and followed three steps:

(1) We presented the REGAIN approach, including examples of grids, and we answered questions from the practitioners.
(2) We asked each practitioner to use the REGAIN approach to capture one to three architectural decisions that practitioners made in their recent work. We did not ask architects to make new decisions. As tool support for REGAIN, we used a web tool called WebGrid [22].
(3) Participants filled out a post-questionnaire, and offered additional feedback in semi-structured interviews.

Regarding ethical concerns, we asked practitioners to omit confidential information from the sessions. We indicated that results were to be used only for research purposes and that participants could withdraw at any time from the sessions or refuse to answer questions. We made audio recordings of the sessions, with the permission of each practitioner. Most sessions were conducted face to face, except for two sessions that took place via Skype.

Overall, we interviewed 16 practitioners from 14 different organizations. Table 5 in the Appendix section shows details on the practitioners and the sessions. Overall, practitioners had an average of 9 years of experience in the software engineering industry. In total, practitioners used the REGAIN approach to capture 24 architectural decisions that the practitioners made in their industry projects from various application domains. As part of the sessions, we showed practitioners the REGAIN output for their decisions. This way practitioners could get a clear idea on what to expect from using the REGAIN approach on their real-world decisions, and practitioners were able to offer feedback on the REGAIN approach.

To answer RQ1 and RQ2, we analyzed the answers from the post-questionnaire and the transcripts of the audio recordings from the interviews. We used descriptive statistics on the answers in the post-questionnaire. Two of the authors of this paper performed content analysis on the transcripts and assigned one of the following three codes to pieces of content: advantage (RQ1), disadvantage (RQ1), or improvement opportunity (RQ2) for the REGAIN approach.

3.2. *RQ1 — REGAIN advantages and disadvantages*

3.2.1. *Post-questionnaire analysis*

Table 1 shows the statements in the post-questionnaire. Statements ID1 to ID 5 relate to advantages of REGAIN, in terms of helping communicate the rationale of decisions (ID1 and ID5), offering structured explanation and documentation of the

Table 1. The values indicate the average and standard deviation of the indicated ratings for each statement.

ID	Statement	Answers from architects	Answers from students	Difference	p-value
ID1	The output of the session helps explain to a colleague why I took a certain decision.	4.29 0.59	4.14 1.07	0.15 −0.48	96.7%
ID2	The session provides a structured way of explaining architectural decisions.	3.86 0.74	4.14 0.38	−0.28 0.36	33.6%
ID3	The session provides a structured way of documenting architectural decisions.	4.07 0.59	3.57 0.53	0.50 0.06	8.4%
ID4	The session provides a structured way of making architectural decisions.	3.64 0.61	4.00 1.29	−0.36 −0.68	38.6%
ID5	When trying to understand why another architect took a certain decision, I could use such an output.	4.00 0.76	4.29 0.49	−0.29 0.27	44.6%
ID6	I found the session too tiring for the output.	2.21 1.08	2.43 1.13	−0.22 −0.05	64.2%

rationale (ID2, ID3), and supporting structured architectural decision-making (ID4). Statement ID6 allowed architects to indicate the effort required for using REGAIN. Architects indicated their level of agreement with each statements using 1 (strongly disagree), 2 (disagree), 3 (neutral), 4 (agree), or 5 (strongly agree). We notice that architects agreed with the statements about communicating rationale (ID1 and ID5), structured explanation (ID2), documentation (ID3), and making (ID4) of architectural decisions. Finally, practitioners perceived a reasonable amount of spent effort (ID6).

In Table 1, we compare the answers of the industry architects with the answers from the students in a previous study [11], in which students filled out the same post-questionnaire, after using the REGAIN approach.

We notice that there are only small differences on averages and standard deviations between the answers from students and the answers from industry architects (i.e. fifth column in Table 1). We investigate further the differences, by checking if there are any statistical significant differences between students and industry architects. The two samples (i.e. students and practitioners) are independent from each other, as they consist of different persons. We cannot assume that the samples are drawn from a normal distribution, so we use a non-parametric test: the Mann-Whitney U test. The null hypotheses are that *there are no differences between the two samples on the statements ID1 to ID6*. We use SPSS to apply the Mann-Whitney U test, and we obtain the p-values in Table 1. Since all values are higher than 5%, we cannot reject any of the null hypotheses. Therefore, we consider safe to accept that there are no statistically significant differences between the answers from students and the answers from practitioners on the six statements in Table 1.

Overall, the results of the post-questionnaire indicate two points. First, the REGAIN approach provides value to practitioners, in terms of costs (i.e. not too tiring) and benefits (i.e. systematic approach). Second, the REGAIN approach provides value to both inexperienced architects (i.e. students) and industry architects.

3.2.2. *Transcripts content analysis*

Following the content analysis of the interviews transcriptions, we identified the following codes for advantages.

(1) **Decision-making support** — This code refers to helping architects in their decision-making. This code was indicated by 6 out of 16 participants. One participant remarked: "*now I have a more clear view about the choices that have been made and the analysis provided a more clear direction on where to move in the future.*" Another participant said: "*Without such a tool, I don't know how to distinguish between the different possibilities, in a more evident way. This tool provides evident results of what is really the ideal for this situation.*"

(2) **Decision rationale** — This code refers to offering rationale for the architectural decisions, and it was indicated by five participants. One participant said: "*Also to explain your decision to others, I think it would give a good impression, and easy to understand. If someone sees it, maybe he doesn't know the project.*"

(3) **Reasoning support** — This code refers to the architects obtaining a clearer perspective on their decision, and it was indicated by four participants. One of the participants said: "*from my personal view, now I have a clearer view about choices I made.*" Another participant said: "*The nice thing about this tool is it's very simple. If people need a lot of time for criteria, that's because they don't have the criteria clear in their own mind, that's not a drawback of the tool, it's offering you a mirror about your own conceptions and criteria in your design. It could be a painful experience for somebody who is being interviewed. But that's not a drawback of the tool, it's exposing what you have in your mind.*"

(4) **Systematic** — This code refers to how well structured the approach is, and it was indicated by two participants. One of the participants said: "*I like the structured way of thinking about the decision, the questions you asked: are there any additional issues? Are these similar? Are there communalities? I like the approach very much, I think it's very helpful, it also gives the person you ask a clear focus. It really helps me think about how the decision was taken, what we did, what were the most important constraints, it's something like a feedback so i can reflect myself.*"

Following the content analysis, we identified the following codes for disadvantages.

(1) **Insufficient tool support** — This code was indicated by five participants, who complained on various issues of the tool that we used during the sessions. For example, one participant said: "*I think the tool should have more workflow support, you know the next step, but someone using it for the first time might not. The tool should be more user-friendly.*" Another participant remarked: "*Having some open source decision making tool would be very helpful.*"

(2) **Subjectivity** — This code refers to the fact that REGAIN encourages capturing subjective concerns about the alternatives. While useful for making knowledge explicit, subjective concerns might have limited reusability, since they express personal perspectives that might not be applicable in different situations. This code was indicated by three participants. One of the participants said: "*my criteria might be subjective, to my bias, my experience might not be complete.*" Another participant commented: "*the subjectiveness of the constructs is the big problem for reusability.*"

(3) **Effort** — This code refers to the effort involved for using REGAIN, and this code was indicated by three participants. For example, one participant said: "*people need some training of how this is working. I suggest an e-learning thing to use it.*"

An additional result to the answer for RQ1 emerged from the content analysis: the application context of REGAIN. We obtained the following.

(1) **Project size** — One participant found the approach useful for large projects, rather than for small or medium-sized. Another participant considered the approach useful especially for expensive decisions.
(2) **Domain** — One participant considered the approach useful for enterprise architectural decisions, in addition to software architectural decisions.
(3) **Time** — Regarding when to use the approach, one participant suggested using the approach for ongoing projects, so that decision makers can benefit from using the approach. Four participants suggested using the approach multiple times during the decision lifetime. One of the participant stated: "*It's very nice to do this repetitively, not only for the final choice. For example, after one or two months to apply again these criteria to see where you are moving, alter your changes or criteria, and do it again in two weeks or more, to have an indication on where you are moving compared to your initial goals.*" Two participants suggested using the approach for the early architectural decisions. For example, one participated said: "*It can help a lot in decision making processes. This is very important at the very initial stages of making choices. This is a very good thing to do at the starting point of a project.*"

There are similarities between the feedback from practitioners and the feedback from students in our previous study [11]. Both practitioners and students indicated the same advantages: *systematic*, *insight*, *reflective*, and *decision-making support*. In addition, practitioners and students indicated the same disadvantages: *learning curve*, *tool support*, and *time consuming*. Moreover, practitioners and students indicated similar application context for REGAIN: large projects and early architectural decisions. Overall, the findings from the interviews with practitioners confirm our previous findings from [11].

3.3. *RQ2 — REGAIN improvement opportunities*

Following the content analysis, we identified the following codes for REGAIN improvement opportunities.

(1) **Concerns prioritization** — This code refers to specifying priorities for concerns. Ten participants indicated this code. One of them said: "*Adding in the weights would make it even more precise. That would make it very useful, this is extremely important to me, because in a specific project weights can vary a lot.*"
(2) **Group decisions** — This code refers to using REGAIN for group architectural decisions. Ten participants indicated this code. One of them said: "*I should find a way to do this together [with others], but that will become a mess: sitting around one screen. Maybe you should have an online version where each fills it in, then results are pulled, you get an outcome like this, but averaged for the whole team.*"

(3) **Decision reuse** — This code refers to reusing previous items from REGAIN outputs (e.g. alternatives, concerns) for documenting new architectural decisions. Five participants indicated this code. One of them stated: "*I think it would be easier to reuse it, because you know what kind of criteria you must take into account when making such decisions, and different projects might have few other criteria. That way you can definitely reuse it: this kind of decisions are made in every project, every java project: what kind of UI are we using? So, some of these alternatives would be reusable.*"

(4) **Interpretation assistance** — This code refers to offering guidelines for interpreting the output of cluster analysis, so that architects can interpret the output with the help of the REGAIN tool, without help from another person. Five participants indicated this code. One of them stated: "*perhaps there should be some idea about what the differences, now it's between 64 and 70%: what does that mean? Is that big or small? This can help the reader: if it's less than 10% it's quite similar. Now, it doesn't say anything to me now.*"

(5) **Confidence factors for ratings** — This code refers to indicating confidence factors to ratings, which would express the architect's level of confidence in each assigned rating (e.g. if the architect has a vague idea that a certain alternative has high popularity, then the architect assigns a rating and a low confidence score for the rating). Capturing confidence factors helps the architect document his level of knowledge about the various alternatives. In the study, four participants indicated this code. One of them stated: "*many criteria cannot be assessed from the very beginning of a project's lifecycle; they might reveal themselves at the testing phase. One could have a criterion and give a rating according to estimations, but the real rating could be assigned only when there are some test results in hand.*"

(6) **Optional extra explanations** — Two participants indicated this code, which refers to having the possibility of adding optionally more details to explain the meaning of some concerns or ratings. One participant said: "*What I meant was adding a bit more explanations about what the two poles mean. Clicking on text gives more explanations. Maybe give the user an option so that when scoring, to type why he chose that number. I would like that purely optional, so that people don't need to type a lot. Sometimes, some decisions need to be elaborated more.*"

(7) **Sensitivity analysis** — This code refers to the possibility of analyzing the impact on REGAIN output when changing ratings for a certain concern. Two participants indicated this code. One of them said: "*You should do something on price. What is the influence on the concern? Some price sensitivity thing would be useful.*"

(8) **More types of rating scales** — One participant indicated the need to have more types of rating scales, depending on the nature of the concerns.

(9) **Decision dependency** — One participant indicated the need to consider dependencies among architectural decisions, which refers to offering support for the analysis of not only one architectural decision, but for analyzing sets of

decisions that depend on each other (e.g. choice of web framework and operating system).

3.4. *Discussion*

Our two previous studies with students [11, 12] indicated that REGAIN helps reduce architectural knowledge vaporization. The next step was to investigate the industrial applicability of REGAIN, with the goal of obtaining feedback from practitioners. We formulated research questions on advantages/disadvantages and improvement opportunities for REGAIN, and conducted an interview study to answer the research questions.

The interview study brings empirical evidence on the advantages and disadvantages that practitioners can expect when using REGAIN to make and capture architectural decisions. The results indicate that REGAIN is a systematic approach that offers practitioners important advantages: decision-making support, documentation of decision rationale, and reasoning support. However, REGAIN has some disadvantages: insufficient tool support, subjectivity and required effort. Still, insufficient tool support seems to be the most important disadvantage, since five practitioners indicated it in the study. Another disadvantage, subjectivity (indicated by three practitioners) is a minor disadvantage as it affects only partially the future reuse of captured decisions, since subjective concerns might be less applicable across multiple architects. The effort disadvantage is the expected cost for using any systematic approach. Furthermore, this study shows improvement opportunities for REGAIN, that reduce the disadvantages, and further help practitioners in their activities.

An interesting result is that this study indicates similarity between feedback from students [11, 12] and feedback from practitioners. The similarity suggests that students might be an acceptable proxy for practitioners, in the sense that results from empirical evaluations with students might be applicable to practitioners. Following this study, we agree with Tichy's perspective that students differ marginally from practitioners, except for knowledge of domain-specific areas, large scale systems, and organizations [23]. Tichy's perspective encourages broader usage of students in empirical evaluations, which makes much sense for the software architecture community, because studies with industry architects are very difficult to conduct, due to architects' busy schedules. If more researchers confirm the applicability of results from studies with students to practitioners, within the limits indicated by Tichy, then the software architecture community will be able to produce results with high statistical significance, by getting access to more study participants (i.e. students, instead of practitioners).

The following research directions emerge from this study. First, we identified improvements to REGAIN, such as concerns prioritization, group decisions, support for decision reuse, and addressing decision dependencies. Since concerns prioritization was the most demanded improvement (see Sec. 3.3) by practitioners, we present our efforts to add this improvement in Sec. 4. We will address the other improvements in future work. Finally, insufficient tool support was the most mentioned disadvantage of REGAIN (see Sec. 3.2.2), therefore we implemented user-friendly, open source tool support for REGAIN, which we discuss in Sec. 4.5.1.

4. Phase 3 — Investigate Prioritization Approaches

Architects indicated the need to prioritize concerns for REGAIN, as presented in Sec. 3. Furthermore, priorities of concerns are part of architectural knowledge. For example, if the priority of security is higher than the priority of performance, then this difference in priorities explains why a more secure, but slower alternative for a database system was selected. If these priorities are not captured, then such architectural knowledge is lost, resulting in higher evolution costs. Therefore, it is of paramount importance to capture priorities of concerns in the REGAIN approach.

To identify previous work on prioritization of concerns for REGAIN, we surveyed existing literature on the Repertory Grid technique. We found that some Repertory Grid tools include features to capture priorities of concerns [22, 24]. Shaw and McKnight describe a basic prioritization approach: ranking concerns from one to ten based on their importance [25]. We could not identify any other study on prioritization approaches for the Repertory Grid technique, in particular for prioritizing concerns for architectural decisions.

We chose to investigate prioritization approaches that are already established and accepted in the software architecture community. In Sec. 6, we mention other available prioritization approaches (i.e. ordinal scale approaches) that we considered, but decided not to use with REGAIN. A recent survey on architectural decision-making techniques [26] indicates that two techniques [27, 28] (one of which is CBAM [27]) use the hundred-dollar approach for prioritization, while three other decision-making techniques [29–31] use the pairwise-comparisons approach. In addition, other researchers also use pairwise-comparisons for prioritization in architectural decision-making [32]. Since these two approaches are already established and accepted in the software architecture community, we started to investigate which of them to use with REGAIN.

The hundred-dollar approach requires decision makers to assign a value between 0 and 100 to items that need to be prioritized, so that the sum of all values is 100. The pairwise-comparisons approach (or analytic hierarchy process) requires decision makers to compare all possible pairs of concerns. This helps identify prioritization inconsistencies (e.g. A is more important than B, B is more important than C, but A is as important as C) [33]. Since the hundred-dollar and pairwise-comparisons approaches are already used by other architectural decision-making techniques, this indicates that these two prioritization approaches are familiar and effective to architects, at least to some extent. So, we considered using one of them for improving the REGAIN approach, and we set out to find which one to select.

No study compares the hundred-dollar and pairwise-comparisons approaches for making architectural decisions, so we did not know which one to use with REGAIN.

For choosing between the two prioritization approaches, we considered several factors. First, the performance (in terms of time) of a prioritization approach is important, because architects have busy schedules and they prefer prioritization approaches that take less time. Second, the perception of architects (e.g. learnability) about a prioritization approach influences their decision to use or not to use the approach. Third, architects want to capture accurate priorities, which reflect their perspectives. To summarize: The objective of the study in Phase 3 is to *analyze* the hundred-dollar and pairwise comparisons approaches *for the purpose of* comparison *with respect to* their performance, users' perception, and impact on REGAIN analysis output, *from the viewpoint of* architects, *in the context of* deciding which prioritization approach to add to the REGAIN approach. To achieve this research objective, we performed an experiment with the two prioritization approaches.

Next, we describe participants in the experiment, experimental materials, tasks performed by participants, hypotheses, experimental parameters, design, as well as the results of the experiment.

4.1. *Participants*

Participants in the experiment were graduate students who took the Software Architecture course at the University of Groningen in the Netherlands. As part of the course, students worked in groups of six over a period of ten weeks to architect a home automation system that would interact with the Smart Grid to trade electrical power, and manage energy consumption of home appliances. For the experiment, we prepared tasks for which students had enough training, to eliminate the need for professionals-only knowledge. Tichy argues that computer science graduate students differ marginally from professionals, in the sense that professionals might be more knowledgeable of domain-specific areas, large scale systems, and organizations [23]. Therefore, if a study does not require such extra knowledge from graduate students, then study results are applicable to professionals. Thus, we minimized knowledge requirements about domain-specific areas, large systems and organizations, to make our study results applicable to professionals.

We addressed the ethical aspects of conducting the experiment as part of a course [34] by using a checklist for integrating research and teaching goals [35, 36]:

(1) **Ensure adequate integration of the study into the course topics** — The software architecture course stressed the importance of architectural decisions. During the experiment, students learnt about using the REGAIN approach to capture architectural decisions.

(2) **Integrate the study timeline with the course schedule** — The course schedule included an optional seminar session in week four, in which we conducted our experiment. In week four, students already had made a few architectural decisions for their course project. However, they had to make and capture more architectural decisions. Thus, they could benefit from REGAIN in their course project.

(3) **Obtain subjects' permission for their participation in the study** — We announced the optional seminar session at the beginning of the course. We informed students that the seminar would address advanced software architecture topics. By showing up for the study, students indicated implicitly their consent to participate in the study. We specified clearly that participation or performance in the seminar had no impact on grades.
(4) **Plan follow-up activities** — To increase the educational value for students, we scheduled debriefing sessions with groups of participants, within three weeks after the study. We prepared individual packages, so that students could see their own results. In each half-hour debriefing session, we presented the study details, preliminary results, and we answered questions from students.

Of the 36 graduate students enrolled in the software architecture course, 30 participated voluntarily in our experiment. 23 participants were Dutch, two Indonesian, one Icelander, one Indian, one German, and one Chinese. All participants had bachelor's degrees, in computer science or informatics (20), industrial engineering management (8), industrial automation (1) or Mathematics (1). Participants had an average of 3.6 years (standard deviation of 2.2) of practical experience in software development, and an average of 1.2 years (standard deviation of 1.3) in software architecture.

4.2. *Experimental materials and tasks*

We prepared the following experimental materials, which are available online at [37].

(1) Instruction about how to perform experimental tasks, including examples.
(2) Two grids that described two architectural decisions that students had to make for their course project: choice of user-interface and choice of storage. The grids included decision topics, decision alternatives and concerns for each of the two topics, so that participants would not have to use the whole REGAIN approach during the experiment, due to time constraints. We derived decision topics, decision alternatives and concerns for the two grids from project reports from previous years. Table 6 in the Appendix section shows the grid for the user-interface architectural decisions.
(3) Four prioritization forms: two types of prioritization forms for both grids. For the hundred-dollar approach, participants had to fill out amounts for each concern in a grid, depending on their perceived importance, with the constraint that sum is 100. For pairwise-comparisons, participants had to fill out comparisons among all pairs of concerns (e.g. moderately more, extremely less important). Both types of forms also asked participants to fill in the time when they started the prioritization task and when they completed it.
(4) A post-questionnaire on the background of participants, questions about how participants perceived each prioritization approach, and a section for general feedback on the experiment.

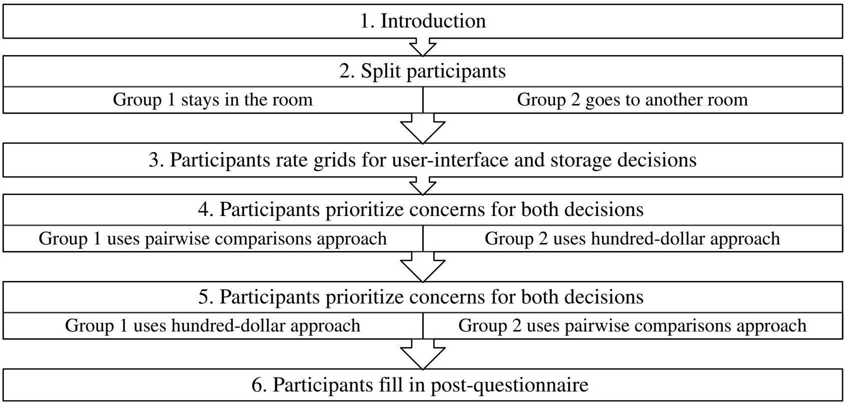

1. Introduction
2. Split participants
Group 1 stays in the room
Group 2 goes to another room
3. Participants rate grids for user-interface and storage decisions
4. Participants prioritize concerns for both decisions
Group 1 uses pairwise comparisons approach
Group 2 uses hundred-dollar approach
5. Participants prioritize concerns for both decisions
Group 1 uses hundred-dollar approach
Group 2 uses pairwise comparisons approach
6. Participants fill in post-questionnaire

Fig. 4. The experimental process had some different steps for Group 1 (left) and Group 2 (right).

Figure 4 shows the steps for executing the experiment, which took place in one session. All students gathered in the same room. In the first step, we presented the plan for the session, we gave a brief introduction on the key concepts for the session, and an overview of tasks to be performed by the students. In step 2, half of the participants went to a different room, accompanied by one research assistant. In step 3, all participants rated alternatives against concerns for the two architectural decisions, from their personal perspectives, using their knowledge on their project. In steps 4 and 5, participants prioritized concerns for both decisions with the hundred-dollar and pairwise comparisons approaches. We collected the paper forms for step 4, before handing over the forms needed for step 5. In the final step, participants filled in the post-questionnaire.

Throughout the session, we encouraged participants to ask for clarifications about their tasks. Furthermore, we placed no time restriction on tasks. This ensured that participants understood their tasks and were comfortable with their tasks.

4.3. *Hypotheses and Variables*

We present the hypotheses on the prioritization approaches, regarding their impact on performance, users' perceptions, and REGAIN output.

4.3.1. *Performance*

We consider two dimensions for evaluating performance of prioritization techniques: required time to perform prioritization using an approach, and scalability of the approach. Scalability concerns the variation of required time for a prioritization approach with the number of concerns to be prioritized. We define the following scalability ratio metric to measure scalability.

$$rs = \frac{t_a}{t_b} * \frac{b}{a} \tag{1}$$

For this metric, a and b are the number of prioritized concerns, and a is smaller than b. A participant needs a time period t_a to prioritize a concerns and a time period of t_b to prioritize b concerns. rs indicates the variation of the time required to prioritize concerns when the number of concerns increases. For example, if rs equals 1, then doubling the number of concerns means that at least twice as much time is needed to prioritize them (linear scalability). If rs is smaller than 1, rs indicates negative scalability. If rs is larger than 1, then doubling the number of concerns needs at least less than half the time (positive scalability).

To evaluate the performance of the hundred-dollar and pairwise comparisons approaches, we test the following null hypotheses.

H_{a0}: Participants need the same amount of time for prioritization.
H_{b0}: The hundred-dollar and pairwise comparisons approaches have the same scalability ratios.

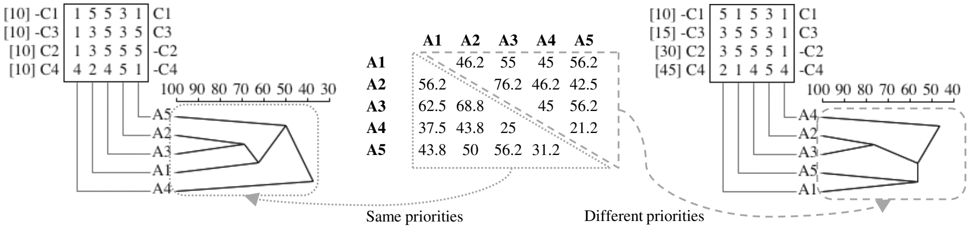
4.3.2. *Users' perceptions*

We operationalize users' perceptions in terms of ease of use, ease to learn and attractiveness of a prioritization approach. The post-questionnaire and the results are presented in the analysis section, in Table 3. We test the following null hypotheses for users' perceptions.

H_{c0}: Participants perceive the hundred-dollar ($100) as equally easy to use compared to the pairwise comparisons (PWC) approach.
H_{d0}: Participants perceive the hundred-dollar ($100) as equally easy to learn compared to the pairwise comparisons (PWC) approach.
H_{e0}: Participants perceive the hundred-dollar ($100) as equally attractive compared to the pairwise comparisons (PWC) approach.

4.3.3. *REGAIN output*

Priorities of concerns influence REGAIN output, which is the result of cluster analysis on the grid with the architectural decision. Figure 5 shows an example of the impact of prioritization approaches on the REGAIN output. The example uses two grids with identical decision alternatives (A1 to A5), concerns ((C1, -C1) to (C4, -C4)) and ratings. However, the concerns in the two grids have different priorities. All concerns in the left grid have a priority of 10. In contrast, the concerns in the right grid have various priorities (i.e. 10, 15, 30, and 45). In Fig. 5, the lower triangular matrix shows percentages of similarities among decision alternatives A1 to A5 (left grid), as calculated by the WebGrid tool. The upper triangular matrix shows percentages of similarities among A1 to A5 for the right grid. For example, the similarity between A2 and A3 is 68.8 for the left grid, and 76.2 for the right grid.



	A1	A2	A3	A4	A5
A1		46.2	55	45	56.2
A2	56.2		76.2	46.2	42.5
A3	62.5	68.8		45	56.2
A4	37.5	43.8	25		21.2
A5	43.8	50	56.2	31.2	

Fig. 5. Percentages of similarities between pairs of alternatives vary by modifying priorities of concerns.

We compare clusters based on the distance between the similarity matrices from which the clusters are derived. The distance between the similarity matrices is calculated as follows:

$$\text{Distance}_{\text{SP,PWC}} = \sum_{i=1}^{N} \sum_{j=1}^{i} \left| \text{SP}_{i,j} - \text{PWC}_{i,j} \right| \qquad (2)$$

In Eq. (2), N is the number of alternatives. SP is the similarity matrix of alternatives, calculated assuming that all concerns have the same priority. PWC is the similarity matrix of alternatives, calculated with the priorities obtained from using the pairwise comparisons approach. Using the above metric, we define the following hypothesis.

H_{f0}: $\text{Distance}_{\text{SP,PWC}} = \text{Distance}_{\text{SP,\$100}} = \text{Distance}_{\text{\$100,PWC}}$
H_{f1}: $\text{Distance}_{\text{SP,PWC}} <> \text{Distance}_{\text{SP,\$100}} <> \text{Distance}_{\text{\$100,PWC}} <> \text{Distance}_{\text{SP,PWC}}$

4.3.4. *Summary*

Overall, in this experiment we investigate how prioritization approaches influence performance, users' perceptions, and REGAIN output. We summarize the independent and dependent variables in Table 2. The first column shows variables and corresponding hypotheses. The last column shows how the hypotheses will be tested.

Table 2. Independent and dependent variables.

Variable	Hypothesis	Type	Scale	Unit	Range	Testing procedure
Prioritization approach	NA	Independent	Nominal	NA	PWC, \$100	NA
Needed time	H_a	Dependent	Ratio	Seconds	> 0	Mann-Whitney test
Scalability	H_b	Dependent	Interval	NA	> 0	Mann-Whitney test
Ease to use	H_c	Dependent	Nominal	NA	PWC, \$100	Binomial test
Ease to learn	H_d	Dependent	Nominal	NA	PWC, \$100	Binomial test
Attractiveness	H_e	Dependent	Nominal	NA	PWC, \$100	Binomial test
Distance between similarity matrices	H_f	Dependent	Ratio	Percent	$>= 0$	Kruskal-Wallis, Mann-Whitney tests

4.4. *Experiment design and results*

This experiment used a randomized, paired comparison design, with one factor and two treatments [38]. The factor was the prioritization approach. Treatments were the hundred-dollar and pairwise-comparisons approaches. Since each subject used both prioritization approaches, the order effect was a validity threat, as results might be confounded by using one prioritization approach before the other. Also, participants might experience fatigue or boredom after using an approach. Therefore, we used counterbalancing in our design, by randomly assigning participants to two similar groups. The two groups used the prioritization approaches in different orders, as shown in Fig. 4: participants in Group 1 used pairwise-comparisons, and afterwards they used the hundred-dollar approach. However, participants in Group 2 used the approaches in the reverse order.

To increase validity, we exclude data points from subjects who have a consistency ratio larger than 20% in the pairwise comparisons approach. The rationale is that such data points indicate inconsistencies in the pairwise comparisons (e.g. an inconsistency exists if x is more important than y, y more important than z, but z is more important than x) [39].

4.4.1. *Results on performance*

Regarding required time, the histograms in Fig. 6 present the frequency of time intervals needed by participants to apply each prioritization approach, for both decisions. We note that the user-interface decision (left side) had a maximum time of around five minutes, while the storage decision (right side) had a maximum time of around eleven minutes. Furthermore, the histograms indicate the distribution of the time intervals. For example, the leftmost histogram indicates that most participants (i.e. eleven) needed between 33 and 66 seconds to prioritize concerns for the user-interface decision using the hundred-dollar approach.

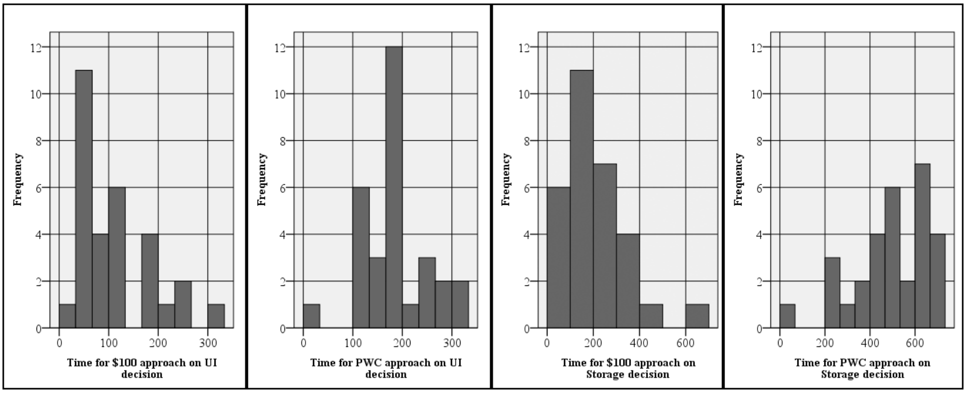


Fig. 6. Histograms with required time (in seconds) to complete prioritization for the user-interface (left) and storage (right) decisions using the two prioritization approaches ($100 and PWC).

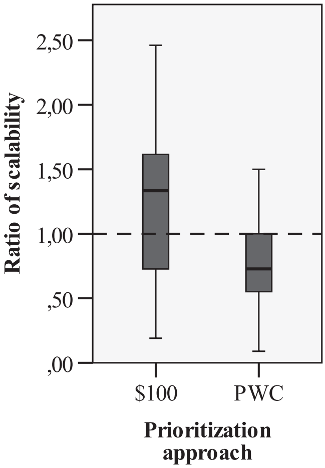


Fig. 7. Boxplots with ratios of scalability (rs) for $100 and PWC prioritization approaches.

We use the Mann-Whitney U Test to test the null hypothesis that both approaches need the same amount of time (H_{a0}) for both decisions. The p-value for both tests is 0.000, so we reject the null hypothesis and we accept the alternative hypothesis that the required time for $100 differs from the required time for PWC. Based on histograms in Fig. 6, we conclude that PWC needs more time than $100.

Regarding scalability of the prioritization approaches, we calculate rs for the two prioritization approaches as outlined in Eq. (1), using the amounts of time for the UI and Storage decisions (t_a and t_b), which have four and eight concerns ($a = 4$, $b = 8$). We remove two outliers with ratios of 6 and 18, and we display boxplots of rs in Fig. 7. Most ratios for the hundred-dollar test indicate positive scalability, while ratios for pairwise-comparisons indicate negative scalability.

To gain more detailed insights into these findings and to check for neutral scalability, we used the Mann-Whitney U test. However, neither hundred-dollar test, nor pairwise comparisons scale neutrally ($p = 0.043$ and $p = 0.00$). Therefore, we accept the alternative hypotheses: hundred-dollar test scales positively, and the pairwise-comparisons approach scales negatively.

4.4.2. *Results on users' perceptions*

Table 3 lists questions from the post-questionnaire related to the perception of users, including the frequency of answers to each of these questions. Answers were either $100 or PWC. To test the hypotheses in Table 3, we ran binomial tests on the numbers for $100 and PWC. We obtained the p-values in the last column in Table 3.

Since all p-values are smaller than 0.05, we accept the alternative hypotheses: participants perceive the hundred-dollar test as easier to use, easier to learn and more attractive than the pairwise comparisons approach.

Table 3. Number of answers in the post-questionnaire.

Hypothesis	Post-questionnaire item	\$100	PWC	p-value
Ease to use	Which of the two approaches was easier to use?	23	7	0.003
Ease to learn	Which of the two approaches was easier to learn?	22	7	0.008
Attractiveness	Which of the two approaches was more fun to use?	28	2	< 0.01
Attractiveness	If choosing between the two approaches, which would you prefer to use in your decision making?	24	6	< 0.01

4.4.3. *Results on REGAIN output*

The boxplots in Fig. 8 summarize the data for the 18 valid data points, for the UI and storage decisions. The vertical axis represents the city block distances among each pair of prioritization approaches.

The Kruskal-Wallis test for the UI decision indicates a statistically significant difference between distances (H(2) = 13.635, $p = 0.001$), with a mean rank of 38.64 for SP-PWC, 22.75 for \$100-PWC, and 21.11 for SP-\$100. Therefore, we reject the null hypothesis that distances between similarity matrices are equal (H_{f0}). Repeated Mann-Whitney tests indicate differences between SP-PWC vs. \$100-PWC ($p = 0.006$), and SP-PWC vs. SP-\$100 ($p = 0.000$), but no difference between \$100-PWC vs. SP-\$100 ($p = 0.962$).

For the storage decision, the Kruskal-Wallis test indicates a statistically significant difference between the distances (H(2) = 16.395, $p = 0.000$), with a mean rank

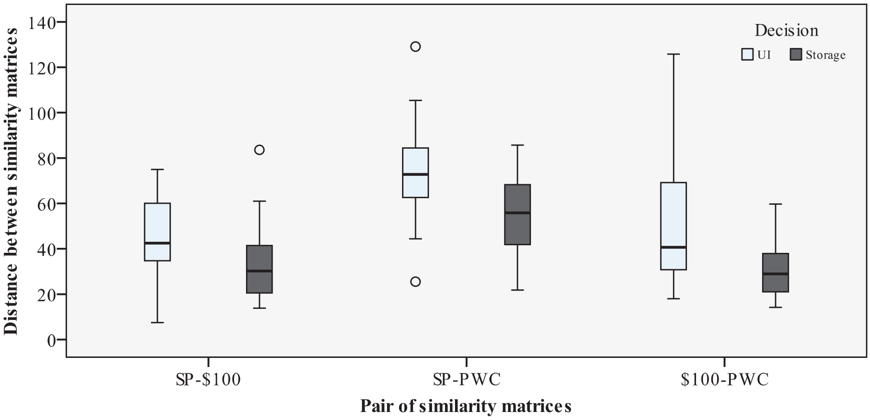


Fig. 8. Distances between similarity matrices for the \$100, PWC, and same priorities (SP) approaches for the two decisions.

of 23.82 for SP-\$100, 41.53 for SP-PWC, and 21.66 for \$100-PWC. Repeated Mann-Whitney tests indicate differences between SP-\$100 vs. SP-PWC ($p = 0.002$), and SP-PWC vs. \$100-PWC ($p = 0.000$). No difference exists between SP-\$100 vs. \$100-PWC ($p = 0.815$).

This suggests that \$100 offers a compromise between using SP and PWC. Different prioritization approaches produce priority values which impact the hierarchical clusters. To evaluate the impact, we compared similarity matrices for grids that use same priorities, pairwise-comparisons, and hundred-dollar prioritization approaches. SP and PWC result in most different matrices. The distance between matrices from using \$100 and SP does not differ from the distance between matrices from using \$100 and PWC.

4.5. *Discussion*

Results from the interview study with practitioners (in Sec. 3) indicated the need to prioritize concerns for the initial REGAIN approach. Also, concerns' priorities are part of architectural knowledge, and implicit priorities increase the risk of architectural knowledge vaporization. Hundred-dollar and pairwise-comparisons are already used in other decision-making approaches [26]. Therefore, we conducted an experiment to compare the hundred-dollar and pairwise-comparisons approaches, to understand which prioritization approach to add to REGAIN.

In the experiment, we investigated performance, users' perceptions, and impact on REGAIN output of the two approaches. On performance, we found out that the hundred-dollar test needs less time than the pairwise-comparisons, and that the hundred-dollar scales better than the pairwise-comparisons approach. Users perceived the hundred-dollar as more attractive, easier to use and to learn than pairwise-comparisons. Finally, we found that prioritization approaches influence REGAIN output, and that the hundred-dollar approach offers a middle ground between the pairwise-comparisons approach and using same priorities for all concerns.

The results on performance and users' perceptions can be explained by the steps of the two approaches. Pairwise-comparisons requires comparisons among each possible pair of concerns, so it has an $O(n^2)$ order of complexity. In contrast, hundred-dollar requires direct indication of priorities. Therefore, if a person has a clear idea about priorities of concerns involved in an architectural decision, then the hundred-dollar approach captures priorities more efficiently than pairwise-comparisons, which also influences users' perceptions. However, if a person does not have a clear idea about the priorities, then pairwise-comparisons might be more efficient, because it offers smaller steps than the hundred-dollar approach. In such situation, the small steps of the pairwise-comparisons approach might provide more value than the hundred-dollar approach.

Following this experiment, we recommend practitioners with little experience in capturing concerns' priorities for their decisions to use the hundred-dollar approach, because first-time users' perceptions favor it over the pairwise-comparisons approach. Also, we recommend practitioners who have a clear idea of the concerns' priorities to use the hundred-dollar approach, because of its better performance. However, we recommend that practitioners who need help clarifying priorities to use the pairwise-comparisons approach, because it offers smaller steps and a consistency ratio.

In practice, the number of items (i.e. concerns) to be prioritized can be larger than the number of items we used in this experiment. This raises questions on the scalability of the hundred-dollar approach for many items to be prioritized. For example, Regnell *et al.* [40] present a study with practitioners who prioritized 58 items. In addition, Berander and Jönsson [41] mention that prioritizing 200 items happens in practice. To cope with the large number of items, researchers propose two approaches. The first approach is to increase the number of points (or dollars), such as using $100,000 for prioritizing the 58 items in [40]. The second approach is to define various levels of abstraction that enable high-level items to be refined into more detailed items, thus creating hierarchies of items in which the hundred-dollar prioritization takes place at each level of the hierarchy. This extension of the hundred-dollar approach is called hierarchical cumulative voting. Berander and Jönsson [41] present hierarchical cumulative voting in detail. Thus, according to [40, 41], the hundred-dollar approach scales better than the pairwise-comparisons approach even for large number of items to be prioritized.

There are tradeoffs for choosing which prioritization approach to add to the REGAIN approach. Although the hundred-dollar approach has better performance and users' perceptions, the hundred-dollar approach does not address potential prioritization judgment errors, as it does not help identify prioritization inconsistencies. This means that the hundred-dollar approach might be less reliable than the pairwise-comparisons approach. However, the experiment results on REGAIN output suggest that the hundred-dollar approach offers results similar to the results from the pairwise-comparisons approach, which indicates the reliability issue is not critical. Therefore, we consider that the hundred-dollar approach offers a good tradeoff for REGAIN concerns' prioritization.

These findings are in line with results from the requirements engineering domain. Karlsson *et al.* found out that the pairwise-comparisons approach is the most trustworthy out of six prioritization approaches, because of its inclusion of a consistency check, which is critical in addressing potential human judgmental errors [42]. Berander and Svahnberg noticed that requirements engineers might prefer the hundred-dollar approach over the pairwise-comparisons approach, because the former is more straightforward [43].

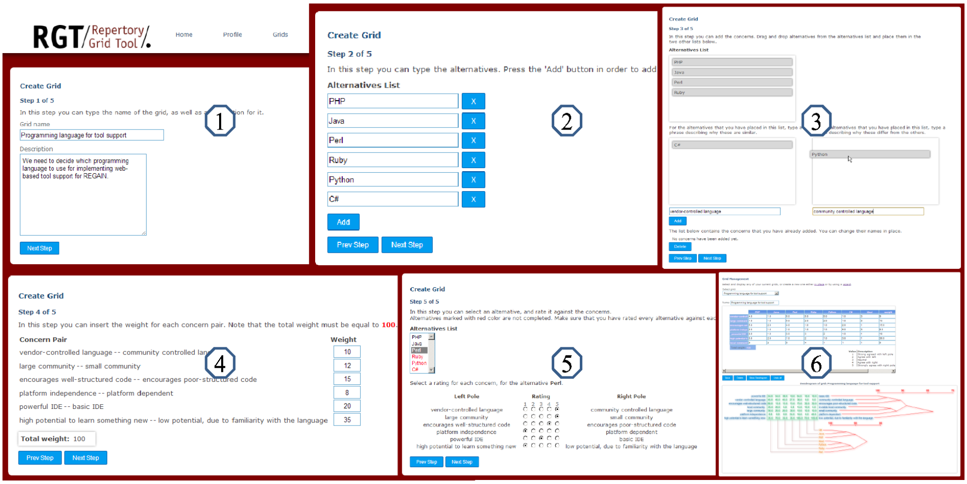
The findings from this experiment are important for several reasons. First, we obtained empirical evidence for the comparison between the hundred-dollar and pairwise-comparisons approaches. The evidence increases our understanding of prioritization approaches for architectural decisions, and removes speculations on the characteristics of the two approaches. Second, since REGAIN is based on the Repertory Grid technique, this experiment encourages other researchers to add a concerns prioritization step when using the Repertory Grid technique for empirical software engineering research (e.g. such as future studies, similar to the Repertory Grid studies in [44]). Third, some results of this study (findings on performance and users' perceptions) can be reused for other architectural decision-making techniques that use the hundred-dollar and pairwise-comparisons approaches [26]. Fourth, based on the experiment results, we add a step with the hundred-dollar approach to the initial REGAIN approach and to its tool support, so that architects can prioritize concerns for their decisions.

4.5.1. *Tool support for REGAIN*

From the industry study with practitioners (in Sec. 3), we learnt that practitioners need tool support for the REGAIN approach, to help architects capture and make architectural decisions. Following the experiment in Sec. 4, we decided that REGAIN tool support should use the hundred-dollar prioritization approach, instead of the pairwise comparisons approach.

We decided to implement a new architectural knowledge tool, rather than reuse an existing one for the following reasons. First, tools for architectural decisions do not include support for capturing decisions with the Repertory Grid technique, so we could not use them with REGAIN. Second, tools for the Repertory Grid technique are intended for generic applications of the Repertory Grid technique, instead of specific application for capturing and making architectural decisions. Third, the feedback from practitioners indicated the need for more user-friendly tool support, compared to the Repertory Grid tool that we used during the interview study.

We implemented an open-source, web-based tool for REGAIN. The tool is deployed at www.repertorygridtool.com. As of August 2013, the tool has 470



Create Grid

Step 1 of 5

In this step you can type the name of the grid, as well as a ion for it.

Grid name: Programming language for tool support

Description: We need to decide which programming language to use for implementing web-based tool support for REGAIN.

Create Grid

Step 2 of 5

In this step you can type the alternatives. Press the 'Add' button in order to add

Alternatives List: PHP, Java, Perl, Ruby, Python, C#

Create Grid

Step 3 of 5

In this step you can add the concerns. Drag and drop alternatives from the alternatives list and place them in the two other lists below.

Alternatives List: PHP, Java, Perl, Ruby

For the alternatives that you have placed in this list, type a phrase describing why these are similar.

C#

vendor-controlled language

alternatives that you have placed in this list, type a describing why these differ from the others.

Python

community controlled language

The list below contains the concerns that you have already added. You can change their names in place.

No concerns have been added yet.

Create Grid

Step 4 of 5

In this step you can insert the weight for each concern pair. Note that the total weight must be equal to **100**.

Concern Pair	Weight
vendor-controlled language -- community controlled lan	10
large community -- small community	12
encourages well-structured code -- encourages poor-structured code	15
platform independence -- platform dependent	8
powerful IDE -- basic IDE	20
high potential to learn something new -- low potential, due to familiarity with the language	35

Total weight: 100

Create Grid

Step 5 of 5

In this step you can select an alternative, and rate it against the concerns.
Alternatives marked with red color are not completed. Make sure that you have rated every alternative against eac

Alternatives List: PHP, Java, Perl, Ruby, Python, C#

Select a rating for each concern, for the alternative **Perl**.

Left Pole	1	2	3	4	5	Right Pole
vendor-controlled language	○	○	○	○	●	community controlled language
large community	○	○	○	○	●	small community
encourages well-structured code	○	○	○	●	○	encourages poor-structured code
platform independence	●	○	○	○	○	platform dependent
powerful IDE	○	○	●	○	○	basic IDE
high potential to learn something new	●	○	○	○	○	low potential, due to familiarity with the language

Grid Management

Select and display any of your current grids, or create a new one either in place or by using a wizard.

Select grid: Programming language for tool support

Name: Programming language for tool support

Value	Description
1	Strong agreed with left pole
2	Agree with left
3	Neutral
4	Agree with right
5	Strongly agree with right pole

Dendrogram of grid: Programming language for tool support

Fig. 9. Six screenshots from the tool with the REGAIN steps and results of cluster analysis.

registered users. The source code of the tool is available at https://github.com/danrg/RGT-tool.

Compatibility, usability, and security were key drivers for the architecture of the REGAIN tool, which is compatible with the recent versions of all major browsers: Internet Explorer, Chrome, and Firefox. Also, the tool uses the jQuery and jQuery UI libraries, for a friendly user-interface and cross-browser compatibility. The Django framework offers multiple security features, such as protection against SQL injections.

Figure 9 shows six screenshots from using the REGAIN tool for capturing an architectural decision we made when choosing the programming language for the REGAIN tool. The screenshots show the steps from the wizard for capturing the decision topic as grid name (1), alternatives (2), concerns (3), prioritization of concerns (4), rating (5), and cluster analysis of the grid (6). In Fig. 10 in the Appendix section, we present a usability feature of the tool for prioritization of concerns.

5. Validity Threats

We discuss internal, construct, conclusion, and external validity threats validity threats for the interview study in Sec. 3 and the experiment in Sec. 4, using the recommendations from Jedlitschka *et al.* [45].

5.1. *Interview study validity threats*

Internal validity refers to causality relationships, which are not applicable for this study, since we do not try to establish a causality relationship.

Construct validity refers to the generalization of study results to the theory behind the study [46]. Construct validity threats may originate from how the study design reflects the studied constructs (e.g. advantages and disadvantages of the REGAIN approach) [46]. To avoid such threats, we allocated enough time to explain the REGAIN approach to the practitioners. We used it on a toy example to make sure that practitioners understood the approach. Also, we selected practitioners who took architectural decisions in the industry, to ensure that they could provide relevant feedback. Moreover, we avoided using a single type of measures (the post-questionnaire), by also using semi-structured interviews, so that practitioners could elaborate on their perspectives.

Another source of construct validity threats is the behavior of the participants and researchers [46]. For example, some participants might change their behavior during a study. We used recommendations from [47] on creating a non-judgmental and comfortable atmosphere during the interviews, so that participants would not change their behavior. Still, researchers might influence subconsciously study results because of their own expectations, which might change researchers' behavior (e.g. extra enthusiasm) [46]. To mitigate this problem, we asked another researcher to conduct the sessions with the architects. This additional researcher was not involved in the previous studies on the REGAIN approach, and had no interest in obtaining either positive or negative feedback on the approach.

Conclusion validity refers to accuracy issues of study conclusions, deriving from statistical and measurement issues [46]. In this study, we used descriptive statistics. Thus, we did not run into issues on low statistical power. Regarding measurement, we used a post-questionnaire validated in a previous study, to ensure reliability of measures, and we used a non-parametric statistical test, which has broader assumptions on the distribution of the data, compared to parametric statistical tests. Furthermore, two researchers were involved in the content analysis of the interviews transcriptions, who agreed on the data interpretation.

External validity refers to the ability of generalizing the study results to industrial practice [46]. To address this validity threat, we recruited architects who worked in a wide variety of domains (see Table 5). Also, the architects used the REGAIN approach on real-world architectural decisions from their industrial practice. To further generalize the results, we needed to make sure that enough architects participated in the study. Unfortunately, architects have busy schedules and limited time to participate in studies, which lead to difficulties in recruiting architects for this study. However, we reached data saturation from the sessions with the sixteen architects, in the sense that the last two session brought no extra insights compared to the previous sessions.

5.2. *Experiment validity threats*

We addressed **internal validity** threats as follows. To avoid the instrumentation validity threat, we piloted the experiment on three persons with software architecting background. We used their feedback to improve the quality of the experimental package. For example, we increased the readability of the paper forms. Another instrumentation validity threat is the use of self-reporting for time needed by participants for the prioritization tasks. We addressed it by reminding participants to record the start and end time for each task during the experiment. Social threats like demoralization of participants influence negatively study results. To avoid that, we made sure that participants received enough directions for the tasks. Furthermore, we asked participants to rate some statements about the session. Results are summarized in Table 4. Overall, subjects considered that directions were clear and they had clear ideas about their tasks. Most subjects were neutral or positive on enjoying the session. We consider these results suggest a good level of

Table 4. Number of session-related answers in the post-questionnaire.

Statement	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
The directions for the exercise were clear	1	—	1	20	8
I had a clear idea on what I had to do in the exercise	—	1	2	20	7
I enjoyed doing the exercise	1	2	12	11	4

subjects' participation and engagement in the exercise, with a positive impact on the internal validity of the experiment.

We addressed the mortality validity threat by integrating the study with the software architecture course, so that subjects joined it voluntarily for the educational value. No participant dropped out from the experiment, although we had a few cases of incomplete data. To avoid selection threats, we randomized the distribution of participants to the two groups. Furthermore, we checked that subjects have similar background on decision-making, e.g. exclude persons with background on prioritization approaches or the repertory grid technique.

We addressed **construct validity** by defining concrete metrics for operationalizing constructs in our experiment, such as the scalability ratio. Another risk was treatments interaction, because each participant used two prioritization approaches. We addressed it partially by randomly assigning participants to two groups that received the treatments in different orders. Social threats for construct validity refer to the altered behavior of subjects when participating in an experiment. We reduced social threats by integrating the study with the course, and eliminating any impact on participants' grades. Additionally, participants did not know our hypotheses, which could have altered their behavior.

We addressed **conclusion validity** by using non-parametric tests, which make fewer assumptions about the distribution of data, such as Mann-Whitney. However, this experiment included around 30 data points, which may negatively impact the statistical power of tests. Furthermore, subjective measures such as perceptions (e.g. ease to use) tend to be less reliable than objective ones (e.g. time). Thus, we piloted the post-questionnaire to ensure that participants understand the items in the post-questionnaire.

We addressed **external validity** threats by ensuring that the experimental tasks did not require a specific level of software architecture experience, and no professional-only knowledge was needed for the experimental tasks. Furthermore, the participants had some practical experience in software development (average of 3.6 years) and software architecture (average of 1.2 years). Kitchenham *et al.* regard students as relatively close to the population of interest, because they are the next generation of software professionals [48]. Finally, increasing commitment level of participants in a study benefits applicability of study results to professionals [34], Since we integrated the experiment timeline with the course schedule, participants in this experiment had a high commitment level. Therefore, we consider the results of the experiments are applicable to software architects of various levels of practical experience.

6. Related Work

The main benefit of REGAIN is reducing the vaporization of architectural knowledge, through capturing (or acquisition) of architectural knowledge. REGAIN is based on the Repertory Grid technique, which has been used successfully for the acquisition of other types of design knowledge outside the software domain. Boose *et al.* [49] used the Repertory Grid technique to capture design knowledge for different types of products from the Boeing company. Hassenzahl and Wessler [50] used the technique to capture design knowledge of industrial products at the Siemens company. Overall these studies indicate the value of the Repertory Grid technique for capturing design knowledge, because the technique encourages capturing the personal perspectives of the designers, and such perspectives are a very important part of design knowledge.

Software architecture related applications of the Repertory Grid technique include the following studies. De Boer and van Vliet [51] used the Repertory Grid technique to compare the mental models of auditors about existing documentation against the semantic structure of the documentation. This was to discovery architectural knowledge in documentation. Shaw and Gaines [52] argue for closer collaboration between the knowledge engineering and software engineering community. Therefore, they present the Repertory Grid technique that helps capturing knowledge on requirements [52]. Niu and Easterbrook [53] use the Repertory Grid technique to capture the knowledge of stakeholders about requirements, and to clarify inconsistencies in how stakeholders use terminology for stating requirements.

Decision-making applications of the Repertory Grid technique include the following studies. Boose [54] presents decision support tools that are based on the Repertory Grid technique, with applicability in various domains. Scheubrein and Zionts [55] propose a decision-making approach based on the Repertory Grid technique, with tool support in the form of an Excel spreadsheet. Castro-Schez *et al.* [56] propose an extension of the Repertory Grid technique for decision-making support for managers.

There is much interest from the software architecture community in architectural decision-making and capturing techniques. Falessi *et al.* [26] indicate three main types of techniques for decision-making:

(1) Naturalistic decision making (or keeping the first available alternative)
(2) Selecting among a finite number of alternatives (or multi-attribute techniques for decision-making)
(3) Selecting among an infinite amount of alternatives (covered by optimization research, such as multi-objective techniques for decision-making)

In practice, the most common type of techniques is selecting among a finite number of alternatives [26]. Falessi *et al.* [26] compare fifteen such architectural decision-making techniques (e.g. CBAM [27], ISO 9126), and notice that each technique involves tradeoffs, as there is no perfect technique. Techniques for capturing architectural decisions include decision views [3, 5], templates [4], and tools (e.g. [6]). Tang *et al.* compare five tools for capturing architectural decisions (e.g. Archium, PAKME) [57].

Prioritization approaches received much attention in the software engineering community, since practitioners need to prioritize concerns, stakeholders, features, and requirements. Prioritization approaches can be grouped in *ordinal scale* and *ratio scale* approaches [41].

For the ordinal scale prioritization approaches, there is an order among prioritized items (e.g. by assigning importance levels), but no arithmetic operations are possible with the values for priorities. Such a prioritization approach is included in RFC 2119 [58], which recommends specific keywords (i.e. *must/should/may*) for prioritization. In addition, IEEE 830-1998 [59] recommends prioritizing by assigning one of following keywords: *essential/conditional/desirable*. The updated version of IEEE 830-1998 is IEEE 29148-2011 [60], which recommends three types of prioritization: either using *shall/will/should/may* keywords, *one-to-five* scale, or *high/medium/low* keywords.

For the ratio scale prioritization approaches, all arithmetic operations are possible, ratios among values and interval sizes are relevant. This means that values obtained from using a ratio scale prioritization approach are more precise than values obtained from ordinal scale prioritization approaches [41]. Pairwise-comparisons and hundred-dollar are the most used ratio scale prioritization approaches [41].

There are important differences between ordinal and ratio scale prioritization approaches, which influenced our choice on which type of approach to use for REGAIN. Although ordinal scale approaches are standardized [58–60] and very common in practice [41], such approaches carry the risk of knowledge vaporization. Results from previous studies [41, 42, 61] indicate that using a ratio scale approach captures more knowledge than using an ordinal scale prioritization approach, such as fine grained differences in the priorities. Since REGAIN aims at capturing knowledge on architectural decisions, ratio scale approaches are better suited for REGAIN, instead of ordinal scale prioritization approaches.

7. Conclusions and Future Work

In this paper, we address three problems. First, REGAIN or the Repertory Grid Technique had not been used with practitioners to make and capture real-world architectural decisions. Second, potential improvements of REGAIN were not known. Third, there was insufficient tool support for REGAIN.

To solve these problems, we conducted an interview study with 16 practitioners. From the interview study, we found advantages, disadvantages, and improvement opportunities for REGAIN. A critical improvement was to prioritize concerns for REGAIN. To select a prioritization approach, we conducted an experiment with 30 graduate students in which we compared the hundred-dollar and pairwise-comparisons approaches.

The results of the experiment indicate that the hundred-dollar approach has better scalability and needs less time than the pairwise-comparisons approach. Also, users' perceptions favored the hundred-dollar approach: participants considered that the hundred-dollar approach was easier to use, learn and more attractive than the pairwise-comparisons approach. Moreover, results suggest that prioritization approaches impact REGAIN output, and that the hundred-dollar approach provides a 'middle ground' between using the same priorities and the pairwise-comparisons approach.

Following the studies with practitioners and students, we implemented tool support for REGAIN. Based on the results of the experiment, the tool uses the hundred-dollar approach for concerns prioritization. The tool is open-source and uses a permissive, business-friendly, open source software license (i.e. MIT license). The development of the tool is ongoing and we add more features to the tool, based on the feedback we received during the interview study and from the tool users.

In future work, we plan to add additional features to the tool support for REGAIN, such as facilitating reuse of concerns based on concern-specific keywords. Also, we plan to evaluate the tool support for REGAIN to understand how much time practitioners might save by using the tool, as well as other improvement points for the tool. Also, we will propose a group architectural decision-making process and tool support based on REGAIN. In addition, we will propose an ISO42010-compliant documentation framework with viewpoints that help reuse architectural knowledge and capture dependencies among architectural decisions.

Appendix

Phase 2 — Additional details

The table below shows details on the architects who participated in the study in Sec. 3.1.

Table 5. Details on the architects who participated in the study about the industrial applicability of REGAIN, and the captured decisions.

ID	Age	Years of experience	No. of captured decisions in session	Application domain of decision
1	30–35	9	2	Bioinformatics
2	25–30	1	1	Home automation
3	30–35	3	1	e-Government
4	25–30	5	2	Bioinformatics
5	35–40	8	2	Bioinformatics
6	30–35	5	2	Healthcare
7	20–25	3	1	Mobile
8	30–35	7	1	Service oriented systems
9	45–50	23	1	Logistics
10	40–45	15	2	Database design
11	45–50	22	3	Telecommunication
12	25–30	2	1	Document-oriented database
13	45–50	14	1	e-Government
14	30–35	7	1	Geographic information system
15	40–45	15	2	Statistics
16	45–50	8	1	Mobile

In Sec. 3.1, we present the sessions that we conducted with the architects. In the third step of the sessions, we asked architects to offer additional feedback on REGAIN through semi-structured interviews. We used the questions below for the semi-structured interviews.

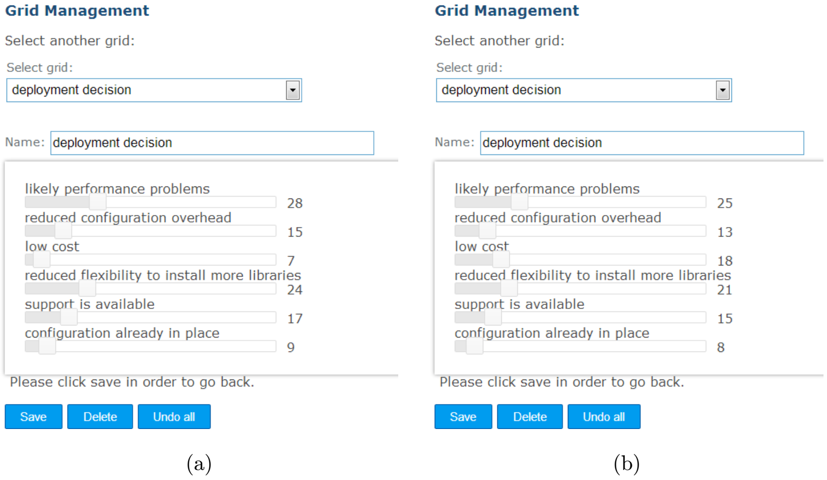
(1) When designing a software system, would you use such an approach to help you with your decision making? If so, why?
(2) In the context of software architecting, what **benefits** do you see for such an approach, and in which activities?
(3) In the context of software architecting, what **drawbacks** do you see for such an approach, and in which activities?
(4) How difficult do you find the approach?
    (a) Which parts were difficult?
    (b) Were the steps and their rational clear?
    (c) What did you like or dislike about it?
    (d) Do you have any suggestions for improvement?
    (e) What do you think about the results of the analysis?

Phase 3 — Additional details

Table 6 shows below one of the decisions (on user-interface), which is part of the experimental materials (see Sec. 4.2).

Table 6. Fixed grid with the decisions on user-interface. A rating of 1 indicates strong agreement with the left pole, and a rating of 5 indicates strong agreement with the right pole, similar to the example in Fig. 3.

UI Decision		**Dedicated touchscreen**	**Web interface**	**Mobile apps**	**Windows application**	***Ideal alternative***	
Concern	**Left pole (1)**						**Right pole (5)**
Learnability	The end user needs a few days to get familiar with the interface						The end user needs half an hour to get familiar with the interface
Implementability	ESH needs a team of 10 people, working for 6 months to implement it						ESH needs a team of 2 people, working for 6 months to implement it
Responsiveness	The interface provides feedback to user's actions anytime between 0.5-5 seconds						The interface provides feedback to user's actions in less than 0.5 second
Interactivity	The interface enables entering text to HPS with the speed of writing an SMS on a phone keyboard						The interface enables entering text to HPS with the speed of keyboard typing



Grid Management

Select another grid:

Select grid: deployment decision

Name: deployment decision

likely performance problems 28
reduced configuration overhead 15
low cost 7
reduced flexibility to install more libraries 24
support is available 17
configuration already in place 9

Please click save in order to go back.

(a)

Grid Management

Select another grid:

Select grid: deployment decision

Name: deployment decision

likely performance problems 25
reduced configuration overhead 13
low cost 18
reduced flexibility to install more libraries 21
support is available 15
configuration already in place 8

Please click save in order to go back.

(b)

Fig. 10. User-friendly prioritization feature: increasing priority of *low cost* concern (a) decreases priorities of other concerns (b).

Tool support — Additional details

We present a usability feature (mentioned in Sec. 4.5.1) for the REGAIN tool. Tool users can specify directly the priorities of concerns, as visible in step 4 in Fig. 9. However, in the interview study in Sec. 3, practitioners stressed the importance of a user-friendly tool. Therefore, we added an additional feature that helps tool users to prioritize concerns in a visual manner, as in Fig. 10. Users can adjust the priorities by dragging the sliders for concerns. If the slider for a concern is dragged to the right, the tool automatically increases the priority of that concern, and decreases proportionally the priorities of all other concerns, so that their total sum remains 100. The screenshots in Fig. 10 illustrate how increasing the priority of the *low cost* concern decreased the other priorities proportionally with their initial priorities.

Acknowledgments

We thank Konstantinos Tselios for his help in conducting the studies, and participants for their efforts to contribute in the studies.

References

1. J. Bosch, Software Architecture: The next step, in *First European Workshop on Software Architecture*, 2004, pp. 194–199.
2. A. Jansen and J. Bosch, Software architecture as a set of architectural design decisions, in *5th Working IEEE/IFIP Conference on Software Architecture*, 2005, pp. 109–120.
3. P. Kruchten, R. Capilla and J. C. Dueñas, The decision view's role in software architecture practice, *IEEE Software* **26** (2009) 36–42.

4. J. Tyree and A. Akerman, Architecture decisions: Demystifying architecture, *IEEE Software* (2005) 19–27.
5. U. van Heesch, P. Avgeriou and R. Hilliard, A documentation framework for architecture decisions, *Journal of Systems and Software* **85** (2012) 795–820.
6. R. Capilla, J. Dueñas and F. Nava, Viability for codifying and documenting architectural design decisions with tool support, *Journal of Software Maintenance and Evolution: Research and Practice*, **22** (2010) 81–119.
7. N. Milton, *Knowledge Acquisition in Practice: A Step-by-step Guide (Decision Engineering)* (Springer, 2007).
8. D. Tofan, Tacit Architectural Knowledge, in *ECSA 2010*, 2010, pp. 8–10.
9. J. H. Boose, Personal construct theory and the transfer of human expertise, in *Proceedings of the National Conference on Artificial Intelligence*, Texas, USA, 1984, pp. 27–33.
10. B. R. Gaines and M. L. G. Shaw, Knowledge acquisition tools based on personal construct psychology, *Knowledge Engineering Review* **8** (1993) 49–85.
11. D. Tofan, M. Galster and P. Avgeriou, Capturing tacit architectural knowledge using the repertory grid technique (NIER Track), in *33rd International Conference on Software Engineering*, 2011, pp. 916–919.
12. D. Tofan, M. Galster and P. Avgeriou, Reducing architectural knowledge vaporization by applying the repertory grid technique, in *5th European Conference on Software Architecture*, Springer, Germany, 2011, pp. 244–251.
13. R. C. De Boer, R. Farenhorst, P. Lago, H. Van Vliet, V. Clerc and A. Jansen, Architectural knowledge: Getting to the core, LNCS Vol. 4880, 2007, p. 197.
14. P. Kruchten, P. Lago and H. van Vliet, Building up and reasoning about architectural knowledge, in *Proceedings of the International Conference on Quality of Software Architectures* (Springer, 2006) pp. 43–58.
15. P. Kruchten, P. Lago, H. van Vliet and T. Wolf, Building up and exploiting architectural knowledge, in *5th Working IEEE/IFIP Conference on Software Architecture*, 2005, pp. 291–292.
16. I. 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE, 2000.
17. P. Kruchten, An ontology of architectural design decisions in software intensive systems, in *2nd Groningen Workshop on Software Variability*, 2004, pp. 54–61.
18. F. B. Tan and M. G. Hunter, The repertory grid technique: A method for the study of cognition in information systems, *MIS Quarterly*, **26** (2002) 39–57.
19. D. Jankowicz, *The Easy Guide to Repertory Grids* (Wiley, 2003).
20. D. Jankowicz, Why does subjectivity make us nervous? Making the tacit explicit, *Journal of Intellectual Capital* **2** (2001) 61–73.
21. M. L. G. Shaw, *On Becoming a Personal Scientist: Interactive Computer Elicitation of Personal Models of the World* (Academic Press, 1980).
22. B. R. Gaines and M. L. G. Shaw, WebGrid Evolution through Four Generations, 1994–2007, 2007, http://pages.cpsc.ucalgary.ca/∼gaines/reports/KBS/WGEvolution/WGEvolution.pdf, last accessed on August 2012.
23. W. F. Tichy, Hints for reviewing empirical work in software engineering, *Empirical Software Engineering Journal* **5** (2000) 309–312.
24. J. Boose, J. Bradshaw, C. Kitto and D. B. Shema, From ETS to Aquinas: Six years of knowledge acquisition tool development, in *Proceedings of the 5th Knowledge Acquisition Workshop*, 1990.
25. M. L. G. Shaw and C. McKnight, *Think Again: Personal Problem-Solving and Decision-Making* (Prentice Hall, 1981).

26. D. Falessi, G. Cantone, R. Kazman and P. Kruchten, Decision-making techniques for software architecture design: A comparative survey, *ACM Computing Surveys* **43** (2011) 1–28.
27. R. Kazman and M. Klein, Quantifying the costs and benefits of architectural decisions, in *Proceedings of the International Conference on Software Engineering*, 2001, pp. 297–306.
28. T. Gilb, *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage* (Butterworth-Heinemann, 2005).
29. A. Andrews, E. Mancebo, P. Runeson and R. France, A framework for design tradeoffs, *Software Quality Journal* **13** (2005) 377–405.
30. T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi and B. Benatallah, A quality-driven systematic approach for architecting distributed software applications, in *Proceedings of the International Conference on Software Engineering*, ACM, New York, USA, 2005, pp. 244–253.
31. M. Svahnberg, C. Wohlin, L. Lundberg and M. Mattsson, A quality-driven decision-support method for identifying software architecture candidates, *International Journal of Software Engineering and Knowledge Engineering* **13** (2003) 547–573.
32. F. H. Jabali, S. M. Sharafi and K. Zamanifar, A quality based method to analyze software architectures, *International Journal of Computer Science* **8** (2011).
33. T. Saaty, How to make a decision: The analytic hierarchy process, *European Journal of Operational Research* **48** (1990) 9–26.
34. P. Berander, Using students as subjects in requirements prioritization, in *Proceedings of the International Symposium on Empirical Software Engineering*, 2004, pp. 167–176.
35. J. C. Carver, L. Jaccheri, S. Morasca and F. Shull, A checklist for integrating student empirical studies with research and teaching goals, *Empirical Software Engineering* **15** (2009) 35–59.
36. M. Galster, D. Tofan and P. Avgeriou, On integrating student empirical software engineering studies with research and teaching goals, in *Proceedings of the Evaluation and Assessment in Software Engineering*, 2012.
37. D. Tofan, Appendix, http://www.cs.rug.nl/∼dan/PrioritizationExperiment/, last accessed August 2012.
38. N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation* (Springer, 2001).
39. J. Karlsson and K. Ryan, A cost-value approach for prioritizing requirements, *IEEE Software* **14** (1997) 67–74.
40. B. Regnell, M. Höst, J. N. Och Dag, P. Beremark and T. Hjelm, An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software, *Requirements Engineering* **6** (2001) 51–62.
41. P. Berander and P. Jönsson, Hierarchical cumulative voting (HCV)-prioritization of requirements in hierarchies, *International Journal of Software* **16** (2006) 819–849.
42. J. Karlsson, C. Wohlin and B. Regnell, An evaluation of methods for prioritizing software requirements, *Information and Software Technology* **39** (1998) 939–947.
43. P. Berander and M. Svahnberg, Evaluating two ways of calculating priorities in requirements hierarchies — An experiment on hierarchical cumulative voting, *Journal of Systems and Software* **82** (2009) 836–850.
44. H. M. Edwards, S. McDonald and S. M. Young, The repertory grid technique: Its place in empirical software engineering research, *Information and Software Technology* **51** (2009) 785–798.
45. A. Jedlitschka, M. Ciolkowski and D. Pfahl, Reporting experiments in software engineering, in F. Shull, J. Singer and D. Sjøberg (Eds.), *Guide to Advanced Empirical Software Engineering* (Springer, 2008), pp. 201–228.

46. C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell and A. Wessln, *Experimentation in Software Engineering* (Springer, 2012).
47. S. E. Hove and B. Anda, Experiences from conducting semi-structured interviews in empirical software engineering research, in *11th IEEE International Software Metrics Symposium*, 2005, pp. 23–23.
48. B. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam and J. Rosenberg, Preliminary guidelines for empirical research in software engineering, *IEEE Transactions on Software Engineering* **28** (2002) 721–734.
49. J. Boose, D. Shema and J. Bradshaw, Design knowledge capture for a corporate memory facility, in *Fifth Conference on Artificial Intelligence for Space Applications*, 1990, pp. 271–280.
50. M. Hassenzahl and R. Wessler, Capturing design space from a user perspective: The repertory grid technique revisited, *International Journal of Human-Computer Interaction* **12** (2000) 441–459.
51. R. C. De Boer and H. van Vliet, Architectural knowledge discovery with latent semantic analysis: Constructing a reading guide for software product audits, *Journal of Systems and Software* **81** (2008) 1456–1469.
52. M. Shaw and B. R. Gaines, Requirements acquisition, *Software Engineering Journal* (1996).
53. N. Niu and S. Easterbrook, So, you think you know others' goals? A repertory grid study, *IEEE Software* **24** (2007) 53–61.
54. J. H. Boose, Using repertory grid-centered knowledge acquisition tools for decision support, *Vol. III: Decision Support and Knowledge* **3** (1989) 211–220.
55. R. Scheubrein and S. Zionts, A problem structuring front end for a multiple criteria decision support system, *Comput. Oper. Res.* **33** (2006) 18–31.
56. J. J. Castro-Schez, L. Jimenez, J. Moreno and L. Rodriguez, Using fuzzy repertory table-based technique for decision support, *Decision Support Systems* **39** (2005) 293–307.
57. A. Tang, P. Avgeriou, A. Jansen, R. Capilla and M. Ali Babar, A comparative study of architecture knowledge management tools, *Journal of Systems and Software* **83** (2010) 352–370.
58. S. Bradner, RFC 2119, 1997.
59. IEEE Std. 830-1998, Recommended Practice for Software Requirements Specifications, IEEE Computer Society, 1998.
60. ISO/IEC/IEEE 29148, Systems and software engineering — Life cycle processes — Requirements engineering, 2011.
61. J. Karlsson, Software requirements prioritizing, in *Proceedings of the Second International Conference on Requirements Engineering*, 1996, pp. 110–116.