Supporting Learning Organisations in Writing Better Requirements Documents Based on Heuristic Critiques

Eric Knauss and Kurt Schneider

Software Engineering Group, Leibniz Universität Hannover, Germany
{eric.knauss,kurt.schneider}@inf.uni-hannover.de

Abstract. Context & motivation: Despite significant advances in requirements engineering (RE) research and practice, software developing organisations still struggle to create requirements documentation in sufficient quality and in a repeatable way. **Question/problem:** The notion of good-enough quality is domain and project specific. Software developing organisations need concepts that i) allow adopting a suitable set of RE methods for their domain and projects and ii) allow improving these methods continuously. **Principal ideas/results:** Automatic analysis of requirements documentation can support a process of organisational learning. Such approaches help improve requirements documents, but can also start a discussion about its desired quality. **Contribution:** We present a learning model based on heuristic critiques. The paper shows how this concept can support learning on both the organisational and individual levels.

Keywords: heuristic critiques, requirements documentation, learning software organisations, experience management.

1 Introduction

Requirements Engineering is a key success factor for software projects. A number of approaches exist to support assessing the quality of software requirements automatically [1–4]. If such approaches identify problems in requirements documents, these documents can be improved in a most efficient way. Still, there remains an important question: What is good requirements quality?

Existing approaches focus on removing ambiguity [4, 5]. But ambiguity is not always bad [6]. Removing ambiguous wording might lead to false precision. False precision is always bad. The notion of *good requirements documentation* is often specific to an organisation or even a project. Automatic checks of requirements documents are even more valuable if they support writing requirements in the specific structure. Therefore, automatic checks of requirements documents need to be adjustable. In this paper we investigate if adjustable automated requirements checkers (= experience based requirements tools) can support organisational learning.

Research Question: Can experience based requirements tools support organisational learning?

Contribution. In this paper we describe a *learning model* based on adjustable automatic checks of requirements documents. We show how organisational and individual learning is supported and that requirements engineers can adjust such checkers to their needs, thus encoding their experiences.

B. Regnell and D. Damian (Eds.): REFSQ 2012, LNCS 7195, pp. 165–171, 2012.

2 Related Work

Requirements are often specified using natural language, if only as an intermediate solution before formal modelling. As natural language is inherently ambiguous [7], several approaches have been proposed to automatically analyse natural language requirements in order to support requirements engineers in creating good requirements specifications [1–4]. Typically, such approaches define a specific quality model first. Then indicators are defined for the quality aspects that can be automatically evaluated, as in the ARM tool by Wilson et al. [1]. Often these indicators are based on simple mechanisms, e.g. keyword lists. Newer approaches leverage sophisticated analysis of natural language, e.g. the search for *under specification* in the QuARS tool [8]. Kof, Lee et al. work on extracting semantics from natural language texts [2, 3] by focusing on the semi automatic extraction of an ontology from a requirements document. Their goal is to identify ambiguities in requirements specifications. Gleich and Kof present a tool that is able to detect a comprehensive set of ambiguities in natural language requirements [4].

Often, the discussion of tools that automatically analyse requirements documentation is limited to the discussion of their recall and precision [5]. In this paper, we use a broader model for requirements analysis tools that allows us to describe their usefulness for supporting continuous improvement and organisational learning in requirements engineering activities. We feel supported in this goal by Gervasi's discussion on why ambiguity is not always bad [6]. He argues that people are able to articulate missing knowledge in ways that are then identified as ambiguities. Removing these ambiguities can only be beneficial, if the underlaying uncertainty is removed.

3 Experience Based Tools and Learning

In this paper, we continue previous work on learning (c.f. [9]). Here, we focus on experience based requirements tools: tools that automatically check requirements, give constructive feedback (i.e. an experience), and can be extended with new experience from its users. For further discussion, we introduce the concept of *heuristic critiques*:

Definition 1: *Heuristic Critique* — Computer based feedback to an activity or work product (e.g. requirements documentation) based on experience. A heuristic critique consists of
- a *heuristic rule* that can be evaluated by a computer,
- a notion of the critique's *criticality* (e.g. info, warning, error),
- a *meaningful and constructive message*.

A heuristic critique represents a single automatic requirements check. Furthermore, it supports organizational learning, when integrated in requirements engineering tools [9]: A heuristic critique is a suitable representation of an experience (defined as (i) an observation, associated with (ii) an emotion and (iii) a conclusion or hypothesis [10]).

Example. A developer *observes* that requirements are misunderstood, because they do not specify who is responsible for an action. The developer is annoyed (*emotion*), and *concludes* that passive voice should be avoided. Based on this experience, a heuristic critique can be created: If a *heuristic rule* detects passive voice, it could give a warning (i.e. medium *criticality*), and ask the user to use active voice and state responsibility *constructive message*.

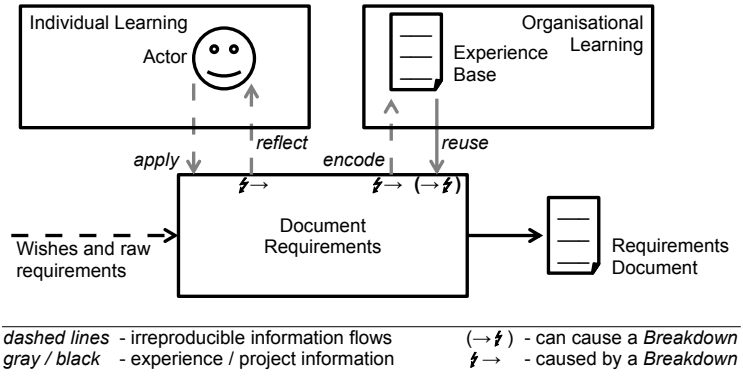

Fig. 1. Learning model: *Heuristic Critiques* stimulate information flows by causing *breakdowns*

We call this concept a *heuristic* critique to emphasize the fact that neither 100% recall nor precision is required. It is more important that these heuristic critiques can be adopted to a specific domain or project environment. By this, it becomes easier to encode new experiences. This is important to support learning. Figure 1 shows two areas of learning, supported by heuristic feedback. Learning occurs on individual and organizational levels during the activity of writing requirements.

Individual Learning: Reflect and Apply. If a heuristic critique fires its warning, the requirements engineer is interrupted in his task. This enables him to *reflect* about the current activity and status, a breakdown occurs. Thus, heuristic critiques facilitate learning through reflection. The requirements engineer might already know how to write good requirements in general. Nevertheless, passive sentences may slip into a specification during periods of intense writing. Reminders and warnings help to *apply* and repeat the knowledge. Thus, they support internalizing abstract knowledge into skills and help writing good requirements.

Organizational Learning: Reuse and Encode. Heuristic critiques allow codifying experiences in a useful way. They support *reuse* of experience, because they enable computers to find situations matching the observation that led to the original experience. Based on the emotion reported, a more or less disruptive message points to potential improvements. Heuristic warnings are not always correct, e.g. an actor could be specified even in a passive sentence. Furthermore, they are not always applicable. If a condition is stated in requirements documentation, use of passive voice is unproblematic. If such a situation is observed during a breakdown, the requirements engineer can refine the heuristic warning and specify that it should not be applied to conditions. Thus, experience is added to the organizations knowledge base. As a by-product, the growing body of codified experience adopts a manageable granularity for an organisation's knowledge base. These advantages have a price: *encoding* experiences as heuristic critiques is more difficult than just writing them down as plain text. In the scope of this paper, we want to concentrate on encoding of new experience.

4 Study: Encoding of New Experience

In this section, we evaluate whether typical requirements engineers are able to encode experiences as heuristic critiques. For the evaluation we need an examplary implementation of an experience based tool. We chose the Heuristic Requirements Assistant (HeRA) [11], a smart use case editor. In HeRA, heuristic critiques can be directly changed by its user during runtime. All users can change the message of the critique or parameters (e.g. keyword lists). In addition, heuristic rules (encoded in Javascript) can be adjusted. All use cases written in HeRA can be accessed from these scripts.

HeRA has been widely used by students in projects at the end of their Bachelors or during their Masters. We consider this group to be representative for our evaluation: (tomorrow's) young professionals with good background knowledge (software engineering, requirements engineering), but limited experience. In our evaluation, we want to investigate whether a representative selection from this target group is able to solve defined tasks under laboratory conditions. We were able to recruit seven volunteers., two of them still in their Bachelors (3rd and 5th year / regular: 3 years).

If a heuristic critique is encoded, it needs to be stored and managed. These rather technical aspects (c.f. [10]) are beyond the scope of this paper. Thus, our research question can be detailed as follows:

Specific Research Question. Can our subjects change existing or create new heuristic critiques? In this evaluation we focus on the heuristic rules, because we consider them most difficult when encoding experiences as heuristic critiques. We approach this question based on the goal question metric paradigm [12]. Accordingly, we have to define beforehand, when we would accept the results to be positive (see Baseline Hypothesis in Table 1).

Evaluation Approach. Our subjects were asked to solve a number of tasks (c.f. Table 1) under supervision within 45 minutes. First, the subjects should show if they were able to understand an intermediate and a complex heuristic rule (Task 1.a). Then, the subjects should change an existing heuristic rule (Task 1.b) and create new rules; a simple, an intermediate, and a complex rule (Task 2). All subjects had a language description with the most important language constructs for the heuristic rules at hand. In addition, they had the data model of use cases in HeRA.

The first part should show our subjects how heuristic rules work. The heuristic rules in this part served as examples for the other tasks. This part was considered part of the instrumentation and not used in the evaluation. The subjects were asked to log their time for completing tasks.

Discussion of Validity. We give a short discussion of the validity of our study to support correct interpretation of our results.

Internal Validity. The most important internal aspect concerns learning effects. It is much more difficult and takes longer to solve a task of a new type. It can be expected that subjects will learn and solve even more difficult subsequent tasks of the same type better and faster. Our evaluation design reflects this aspect by using Task 1.a only for instrumentation. The experiment was conducted in the late afternoon. Many participants

Table 1. Overview of the tasks

Task	Description	Baseline Hypothesis
Task 1.a.1	Describe the goal of a heuristic rule (correct answer: triggers warning, if use case title has more characters than description).	n/a
Task 1.a.2	Describe the goal of a heuristic rule (correct answer: triggers warning, if condition part of an use case extension is empty).	n/a
Task 1.b	Change rule from Task 1.a.2 to trigger a warning if the reference to the extended step is empty.	$> 75\%$ correct < 10 minutes time
Task 2.1	Create a heuristic rule that triggers a warning . . . if the title of a use case is empty.	$> 75\%$ correct < 15 minutes time
Task 2.2	. . . if the main success scenario has less than three or more than 9 steps.	$> 75\%$ correct < 15 minutes time
Task 2.3	. . . if two use cases have the same title.	$> 50\%$ correct < 15 minutes time

* Answers with small errors are counted as 0.5.

had a class before and might have been tired. Task sheet and language description had minor errors. Luckily, these errors could be accounted for during analysis.

Construct Validity. The main construct aspect concerns our baseline hypothesis. Is it valid to conclude from 75% (50%) correct answers under exam conditions that users are able to create correct simple (complex) heuristic rules? Are 10 minutes for changing and 15 minutes for creating heuristic rules short enough to allow users doing this during their workday? Because of the strict evaluation of the answers, we consider such results to be good compared to exams in programming language classes. Analysts could integrate these tasks in their daily work, given they take less than 15 minutes. As opposed to our experiment, analysts would be supported by error messages from a compiler and could directly observe the effects of their rules in HeRA.

Conclusion Validity. Because of the low statistical power, small or medium derivations in the result could be expected in case of replication of the experiment. The specific time of the experiment in the late afternoon after another class could have affected the performance of the subjects negatively in comparison to a replication.

External Validity. For an analyst in industry it might be hard to bring herself to work on heuristic critiques on top of their main tasks. As opposed to our subjects, the analyst has to switch her cognitive context from the current task to the programming of a heuristic rule. We expect the effort for this to be lowest, while the analyst is concerned with quality assurance of requirements documentation.

Results. Figure 2 shows the results of our study. The figure shows the minimal, average, and maximal time it took our subjects to solve the tasks (black). In addition the percentage of correct answers is shown in gray. The working time for Task 1.a.2 is considerably lower then for Task 1.a.1, probably due to learning effects. It took the subjects only 1–2 minutes to change a heuristic rule (a rule they had already understood

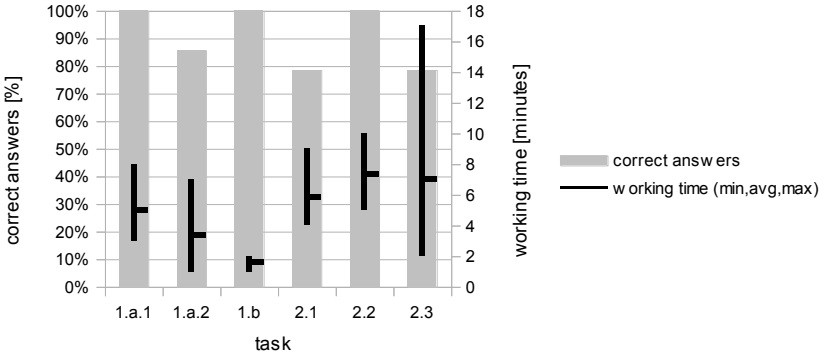

Fig. 2. Results of the experiment. The working time for each task is depicted in black, the percentage of correct answers is shown in gray.

when solving task 1.a.2). The rising difficulty of Tasks 2.1 – 2.3 shows in the maximal working time. The average time goes down from Task 2.2 to Task 2.3. At this time our students were well trained and could easily cope with the complexity. All in all we received 89% correct answers: 100% of the changes at existing heuristic rules were correct and 86% of the newly created heuristic rules. Small changes lasted less than 2 minutes. A new heuristic rule could be created in less than 7 minutes. We expect that these values will even improve if the programming is not performed with pen and paper but with suitable tool support. Thus, this study indicates that users of experience based requirements tools are able to encode new experiences or to adjust existing ones.

5 Discussion and Outlook

Automatic requirements checking has been reported to be beneficial. Yet, automatic requirements checkers are not widely accepted in industry, especially by tool vendors. We argue that existing approaches are either too generic or specific to a too narrow application domain for being widely used. In this paper, we propose to regard automatic requirements checkers as experience based requirements tools. We presented a learning model that helps to tackle effects that follow from the notion of experience based requirements tools. In short, there is more to these tools than just improving requirements documentation. In Section 4 we presented evidence that requirements engineers are able to add new experience to these tools. From a knowledge management perspective, this is an encouraging result. Computational rules that allow to check given documentation allow to formalise knowledge of an organisation in a most useful way. Individuals are confronted with these *heuristic critiques* and are invited to discuss them, based on examples they encounter in their daily work. Our evaluation results suggest that analysts are able to write useful critiques with reasonable effort: Our subjects encoded heuristic critiques that we identified to be useful in less than 7 minutes. Even without special tool support (e.g. wizards, compiler) they did this at a surprisingly low error rate. We conclude that analysts with a computer science background are perfectly capable to express

their experience as heuristic critiques. Based on these results, tools are imaginable that help organisations build an experience base of heuristic critiques specialised on their domain. Our work leads to a number of questions that demand future research. Based on the learning model in Figure 1, we only investigated the *encoding* of new experience. There is still need to gather prove that automated requirements checkers improve requirements documentation. The impact of heuristic critiques on *individual learning* and on the *reuse* of experience should be further investigated.

References

1. Wilson, W.M., Rosenberg, L.H., Hyatt, L.E.: Automated analysis of requirement specifications. In: Proceedings of the 19th International Conference on Software Engineering (ICSE 1997), pp. 161–171. ACM, New York (1997)
2. Kof, L.: Text Analysis for Requirements Engineering. PhD thesis, Technische Universität München, München (2005)
3. Lee, S.W., Muthurajan, D., Gandhi, R.A., Yavagal, D.S., Ahn, G.J.: Building Decision Support Problem Domain Ontology from Natural Language Requirements for Software Assurance. International Journal of Software Engineering and Knowledge Engineering 16(6), 851–884 (2006)
4. Gleich, B., Creighton, O., Kof, L.: Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 218–232. Springer, Heidelberg (2010)
5. Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D.M.: Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requirements Engineering Journal 13(3), 207–239 (2008)
6. Gervasi, V., Zowghi, D.: On the Role of Ambiguity in RE. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 248–254. Springer, Heidelberg (2010)
7. Berry, D., Kamsties, E.: 2. Ambiguity in Requirements Specification. In: Perspectives on Requirements Engineering, pp. 7–44. Kluwer (2004)
8. Fabbrini, F., Fusani, M., Gnesi, S., Lami, G.: An Automatic Quality Evaluation for Natural Language Requirements. In: Proceedings of the Seventh International Workshop on RE: Foundation for Software Quality (REFSQ 2001), Interlaken, Switzerland, pp. 150–164 (2001)
9. Knauss, E., Schneider, K., Stapel, K.: Learning to Write Better Requirements through Heuristic Critiques. In: Proceedings of the 17th IEEE Requirementes Engineering Conference (RE 2009), pp. 387–388. IEEE Computer Society, Atlanta (2009)
10. Schneider, K.: Experience and Knowledge Management in Software Engineering. Springer, Heidelberg (2009)
11. Knauss, E., Lübke, D., Meyer, S.: Feedback-Driven Requirements Engineering: The Heuristic Requirements Assistant. In: Proceedings of the 31st International Conference on Software Engineering (ICSE 2009), Vancouver, Canada, pp. 587–590 (May 2009)
12. van Solingen, R., Berghout, E.: The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. McGraw-Hill Publishing Company (1999)