Feedback-Driven Requirements Engineering: The Heuristic Requirements Assistant

Eric Knauss, Daniel Lübke, Sebastian Meyer
FG Software Engineering, Leibniz Universität Hannover
Welfengarten 1, 30167 Hannover, Germany
{eric.knauss, daniel.luebke, sebastian.meyer}@inf.uni-hannover.de

Abstract

The complexity of today's software systems is constantly increasing. As a result, requirements for these systems become more comprehensive and complicated. In this setting, requirements engineers struggle to capture consistent and complete requirements of high quality. We propose a feedback-centric requirements editor to help analysts controlling the information overload. Our HeRA tool provides analysts with important data from various feedback facilities. The feedback is directly given based on the input to the editor. On the one hand, it is based on heuristic rules, on the other hand, on automatically derived models. Thus, when new requirements are added, the analyst gets important information on how consistent these requirements are with the existing ones.

1 Introduction

Requirements Engineering (RE) deals with elicitation and documentation of the desired results of software projects. Typically, the starting point for these activities is someone's idea to design and build something [4]. A requirements engineer starts with such an idea (i.e. a vision statement) and searches for relevant stakeholders. Typical processes and tools [13] suggest a top-down approach with high-level requirements that are refined step by step until an appropriate description of the system to build is created.

This approach is a good way to cope with the complexity of today's software systems by decomposition into smaller parts. However, one of the inherent problems reported by Gause and Weinberg [4], the different interpretation of the original idea by participants, is not addressed by this approach. Different interpretations are hard to detect when discussing an idea on a high abstraction level. This delays the identification of conflicts until the decomposition has reached a more tangible level that potential users can validate.

Our approach starts following the definition of the vision and the identification of an initial set of stakeholders. The first step is to identify future users and their user goals within the business goals specified by the vision statement. In this paper, we present the Heuristic Requirements Assistant (HeRA), which allows a more bottom-up approach. Its heuristic feedback allows capturing high-quality requirements on user goal level, identifying contradictions to other user's requirements, and aligning user goals with the intended business process quickly. The feedback allows to easily identify topics that need further investigation.

This paper presents HeRA's concepts (Sect. 2), the idea behind each of the included assistants (Sect. 3), and a short overview of evaluation efforts (discussed in more detail in other publications). Section 4 concludes this paper with a discussion of interactions between these parts.

2 Solution

HeRA is based on Fischer's architecture for domain oriented design environments (DODE) [3]. The central part of this architecture is a construction component. In the case of HeRA, requirements are "constructed" using a general-purpose requirements editor, a use case editor, and a glossary editor. These editors allow constructing the domain specific artifacts (i.e. requirements, use cases, and a glossary). HeRA offers two other DODE components, namely the Argumentation Component and the Simulation Component.

Fischer [3] emphasizes the importance of arguing about hints from a domain-oriented design environment. An argumentation component allows users to adhere to warnings (and their respective rules), or to argue against them. Both types of feedback may lead to improved heuristics in the long term.

In contrast, the simulation component gives the requirements author feedback on the effects the current way of modeling Use Cases could have. As we want to use HeRA for

ICSE'09, May 16–24, 2009, Vancouver, Canada
978-1-4244-3452-7/09/$25.00 © 2009 IEEE

the initial documentation of stakeholder wishes we do not have a formal requirements model that could be simulated. However, we can derive certain models that give additional information about the requirements being documented. In this paper, we present three examples for such feedback types based on generated models (UML Use Case Diagrams, EPC Business Processes, and Use Case Points Estimations). Figure 1 gives an overview of the structure of HeRA's components.

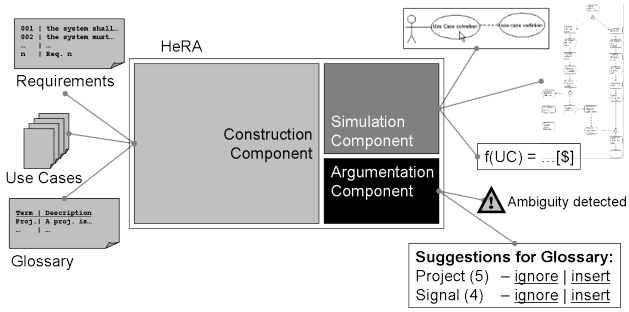

Figure 1. Structure of the HeRA Tool.

Computer-based feedback on requirements documents has often been reported to enhance the quality of requirements specifications. Most of these approaches are focused on the automatic evaluation of software requirements specifications' Quality [2, 17, 11]. As opposed to these approaches we do not analyse the requirements after completion. Instead, we aim at constructively enhancing the knowledge about the system to construct during creation of the specification.

The approach to RE we had in mind when we designed HeRA was to start interviewing future users of the system under development (SUD). Each user's interaction with the SUD is captured in use cases (based on templates suggested by Cockburn [1]). HeRA was designed to support the requirements engineer with heuristic feedback: It analyzes the input and warns the user if it detects ambiguities or incomplete specifications. Furthermore, HeRA generates UML Use Case Diagrams that show how the current user goals relate to the business goal. If needed, a glossary assistant can be used to ensure consistent use of important terms. Another assistant integrated in HeRA is able to generate a visualization of the implied process by concatenating use cases based on post- and preconditions. This permits determining whether the current use case is well-aligned to the global process. On demand, HeRA computes use case points and displays an effort estimation associated with the use cases. All of these perspectives are derived *while* the use case is being written. In this way, the author gets immediate feedback on the input and may improve it. We have successfully applied HeRA during interviews and workshops. Thereby, we achieve the following:

1. Elicitation of user goals on a level of detail that allows for the identification of conflicts,
2. Discussion whether the current user goals fit into the underlying business goal and the other user goals already documented (based on visualization as UML Use Case and process models),
3. Discussion of important terms and identification of conflicting interpretations of these terms,
4. Discussion of prioritization and project constraints based on the use case points.

The direct feedback of HeRA allows starting with elicitation of user goals and detection of inconsistencies and conflicts very early by applying computer-generated feedback.

3 HeRA Components

3.1 Critiques

HeRA's first argumentation component facilitates computer-based critiques. Requirements experts can use HeRA mechanisms to codify their experiences from earlier projects. As a rule set in HeRA, those heuristics will support future requirements engineers with documenting requirements: HeRA applies these rules to the input in the construction component and analyses it. Based on this analysis, HeRA's argumentation component can now give context sensitive feedback on the input.

In HeRA, heuristic rules are defined in JavaScript and have access to the data model of the construction component. Wizards allow authors to generate the code for standard rules. In addition, a description and parameters (e.g. certain keywords) can be given. Rules can be changed during runtime and the results become immediately visible. Typical rules include checking for weakwords (e.g. "someone" and "never"), consistency (e.g. each actor is listed as a stakeholder), and structure (e.g. all user-level use cases are referenced in a business-goal-level use case). In this way, heuristic rules reflect the experience that such situations should be avoided.

The argumentation component as instantiated by HeRA critiques triggers breakdowns during an activity like typing requirements [15]. Authors are interrupted and made aware of a problem they may have overseen. They can decide to either fix it, finish their work and return to the warning later, or give feedback to the warning. Feedback results in an improved rule set for the next project. In [6] we compared the effectiveness of HeRA's critiques to analytical quality assurance and evaluated their positive influence on the quality of the specification.

3.2 Glossary

Frequently used terms have a higher potential for introducing misunderstandings in requirements specifications. Misunderstandings often evolve from tacit knowledge, since every reader of the document may have a different definition for a term.

To counter this problem, we support the creation of a glossary containing such terms. The glossary component is implemented as an argumentation component and works in the background while the requirements are written. It presents a proposal list of terms probably relevant for a glossary. This list helps the author to identify problematic words and add them to a glossary. HeRA allows its users to continue their work and to define such terms later.

In [8], we evaluated how the proposals of glossary terms could be enhanced based on experiences. Terms that were added to glossaries in previous projects get a higher priority in the proposal list.

In addition to proposing possible terms for the glossary, HeRA enhances awareness of terms already defined in the glossary. Whenever such a term is used, it is highlighted. A tool-tip provides the definition from the glossary. Thus, authors can easily ensure that terms are used in the way they were defined.

3.3 Generate UML Use Case Model

A reasonable way to give an overview of a set of use cases and their relationships is a UML Use Case Diagram. HeRA provides such a visualization as a simulation component in order to give direct feedback to the authors.

The component generates an UML Use Case and an UML Actor based on the currently edited use case's Main Actor and Title. We also generate UML Use Cases for each use case referenced by the current one. We create an «includes» relationship when it is referenced in the main scenario. We create an «extends» relationship when the use case is referenced in an Extension of the main scenario. We could have inserted the extension's condition but decided against it in favour of a more compact presentation. We create an inheritance relationshi if the use case is referenced as a Technical Variation: The use case serves the same User Goal but is based on a different implementation.

The result is a quick overview of the most relevant information (Main Actor and Use Case Title) and the immediate context of the current Use Case. The diagram can be generated easily – even while a new title is typed in. Use Cases referenced by the current one may easily be reached by a double click on their UML representation. This makes the UML Use Case Diagram a valuable graphical directory.

3.4 Generate EPC

Very often – especially when developing software supporting business processes like, SOA systems – business processes become part of the software specification. Therefore, the documented use cases have to result in a business process and must cover it completely.

From a large set of textual use cases, it is impossible for a human to quickly derive their order and dependencies between them. Consequently, HERA offers a business process visualization as a simulation component. The use cases are joined based on their triggers, preconditions, and success guarantees. If a use case requires a precondition to be fulfilled, it depends on a use case that has this condition as a success guarantee. The complete algorithm for deriving graphical Event-driven Process Chain (EPC) models [12] from tabular use cases is described in [9, 10]. The visualization of use cases as business processes gives feedback on the global context. Requirements engineers can see whether the global control-flow through the use cases is as expected or not. If it is not, there are a number of possible errors that can be spoted this way:

Missing or wrong conditions: The triggers, preconditions, and success guarantees of the use cases may be wrong. This type of error can lead to a wrong understanding of the desired context by the developers and thus to a not completely fitting software. Especially the preconditions can be important if activities have to be performed in a given order or data must be in a defined state.

Missing use cases: If the generated business process contains gaps or preconditions cannot be satisfied, the requirements engineer might have found missing use cases that would bridge the gap or satisfy the conditions. For instance, if a precondition "user is logged in" is specificed, somewhere there has to be a use case titled like "User logs into the system". While in this case the functionality is self-evident, not documenting it imposes unnecessary risks and may lead to wrong effort estimates.

Mismatching between business processes and use cases: If the overall control-flow is not similar to the specified business process, either the business process might contain unactable parts or the use cases are not suitable at all.

The business process generation can also be used to stimulate the discussion about requirements as described in [7].

3.5 Compute Use Case Points

The Use Case Point method [5] allows estimating project costs of a project based on use cases. It resembles the CO-

COMO and Function Point methods, but relies on analysing scenario steps and actors.

The Use Case Point View in HeRA is implemented as a simulation component (Fig. 1), because it allows a *what-if* analysis based on the use case model. Again, HeRA is able to give an estimation very early, even while Use Cases are being written. These estimations are very inaccurate, but can be valuable if requirements authors reflect upon conspicuous estimations. Requirements Engineers can either adjust the use case model (e.g. adjusting abstraction levels of Use Cases between User Goal and Subfunction), or explicitly change project constraints (e.g. experience of project team, technical risk, …).

HeRA's Use Case Point Component helps authors to switch to the measurers perspective. Quantification and evaluation of this part of HeRA is still open and remains as future work. However, Trudel and Abran [16] investigated the value of such an effort measurement perspective for the quality of requirements specifications. In their work, they compare results of requirements specification reviews done by inspectors and measurers. They claim that the combination of both perspectives leads to an increased defect identification in reviews. Therefore, the Use Case Point Estimation to prove valuable.

4 Conclusion

Figure 2 shows the different levels of feedback given by HeRA. Critiques give immediate feedback on the input, comparable to spell checkers. After editing a Use Case, the proposals for potential glossary terms form another feedback loop. A generated UML Use Case Diagram enhances the overview of modeled use cases. If needed, requirements analysts can request even more feedback: EPCs can be generated to show the global process implied by a given set of use cases, and Use Case Point Estimation can show the effort implied by the current way of modeling the use cases. The speed and frequency of feedback drops from the inside to the outside in Fig. 2. We have evaluated the positive effects of these feedback loops in several projects [6, 8, 7, 14].

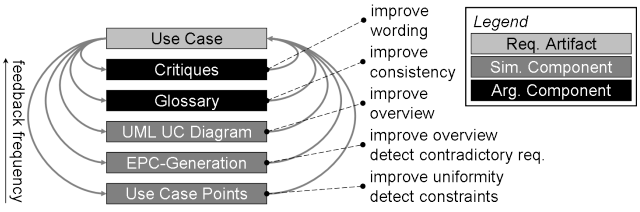

Figure 2. Computer-induced feedback cycles in HeRA

References

[1] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley Professional, January 2000.

[2] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the use of an Automatic Tool. In *SEW '01: Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, page 97, Washington, DC, USA, 2001. IEEE Computer Society.

[3] G. Fischer. Domain-Oriented Design Environments. *Automated Software Engineering*, 1:177–203, 1994.

[4] D. C. Gause and G. M. Weinberg. *Exploring Requirements: Quality Before Design*. Dorset House Publishing Company, Incorporated, 1989.

[5] G. Karner. Resource Estimation for Objectory Projects. *Objectory Systems*, 1993.

[6] E. Knauss and T. Flohr. Managing Requirement Engineering Processes by Adapted Quality Gateways and critique-based RE-Tools. In *Proceedings of Workshop on Measuring Requirements for Project and Product Success*, Palma de Mallorca, Spain, November 2007. in conjunction with the IWSM-Mensura Conference.

[7] E. Knauss and D. Lübke. Using the Friction between Business Processes and Use Cases in SOA Requirements. In *Proceedings of REFS'08*, 2008.

[8] E. Knauss, S. Meyer, and K. Schneider. Recommending Terms for Glossaries: A Computer-Based Approach. In *Proceedings of Workshop on Managing Requirements Knowledge at 16th IEEE RE Conference*, Barcelona, Spain, 2008.

[9] D. Lübke. Transformation of Use Cases to EPC Models. In *Proceedings of the EPK 2006 Workshop, Vienna, Austria*, 2006.

[10] D. Lübke. *An Integrated Approach for Generation in Service-Oriented Architecture Projects*. PhD thesis, Gottfried Wilhelm Leibniz Universität Hannover, 2007.

[11] R. Melchisedech. *Verwaltung und Prüfung natürlichsprachlicher Spezifikationen*. PhD thesis, Fakultät Informatik, Universität Stuttgart, Stuttgart, 2000.

[12] J. Mendling and M. Nüttgens. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). *Information Systems and e-Business Management (ISeB)*, 4(3):245–263, July 2005.

[13] S. Robertson and J. Robertson. *Mastering the Requirements Process,*. Addison-Wesley, 1999.

[14] K. Schneider. Improving Feedback on Requirements through Heuristics. In *Proceedings of 4th World Congress for Software Quality (4WCSQ)*, 2008.

[15] D. A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, 1983.

[16] S. Trudel and A. Abran. Improving the Qualitiy of Functional Requirements by Measuring Their Functional Size. *Proceeding of the International Conferences IWSM, MetriKon, and Mensura*, pages 287–301, 2008.

[17] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt. Automated quality analysis of Natural Language requirement specifications. In *Proceedings of PNSQC Conference*, 1996.