

Preface

This volume contains the papers presented at the 1st International Requirements Engineering Efficiency Workshop (REEW 2011) held on March 31, 2011 in Essen, Germany.

Requirements engineering research has for a long time focused on specification quality, leading to recommendations of how to engineer perfect requirements specifications. Practitioners, however, do not have the time, resources, and interests for overdoing requirements engineering. Rather, many situations call for short-cuts that allow investing effort in those concerns that are critical for success, while reducing effort in other areas where risk is comparably smaller. The social context, smart collaboration processes, and novel ways of looking at the interface between stakeholders and the supplier can be a basis to increase the yield of requirements engineering, while reducing required effort.

The International Requirements Engineering Efficiency Workshop (REEW 2011) aims at initiating, facilitating, and nurturing the discussion on efficient approaches to engineer just good-enough requirements. Requirements engineering is here seen as a means that can be simplified, automated, or combined with other practices to achieve successful systems in an economically efficient manner. REEW 2011 will provide a platform for the community of practitioners and research experts that are interested in efficient and pragmatic approaches to requirements engineering.

The REEW 2011 workshop program featured three talks on requirements engineering efficiency tactics and three talks on requirements engineering efficiency practice.

Each submission was reviewed by 3 program committee members. We would like to thank the program committee for their timely reviews that provided valuable feedback to the authors.

March 16, 2011
Karlskrona and Zurich

Samuel Fricker
Norbert Seyff

Program Committee

Stefan Biff
Samuel Fricker
Vicenzo Gervasi
Tony Gorschek
Paul Gruenbacher
Andrea Herrmann
Sven Krause
Soo Ling Lim
Nazim Madhavji
Anna Perini
Zornitza Racheva
Juha Savolainen
Kurt Schneider
Ken Schwaber
Norbert Seyff
Inge Van De Weerd

TU Wien
Blekinge Institute of Technology
University of Pisa
Blekinge Institute of Technology
Johannes Kepler University Linz
Axivion GmbH
Zühlke Engineering AG
University College London
University of Western Ontario
Fondazione Bruno Kessler
University of Twente
Nokia Research Center
Leibniz Universität Hannover
Scrum.org
University of Zurich
Utrecht University

Table of Contents

Faster from Requirements Documents to System Models: Interactive Semi-Automatic Translation	1
<i>Leonid Kof and Birgit Penzenstadler</i>	
Efficient Requirements Engineering through Feed-Forward	13
<i>Sharmin Ahmed, Muhammad Iftekher Chowdhury, Remo Ferrari and Nazim Madhavji</i>	
Relevant Requirements in Product Line Application Engineering A Foundation to Focus Elicitation	19
<i>Sebastian Adam</i>	
Pragmatic Variability Management in Requirement Specifications with IBM Rational DOORS	25
<i>Ekaterina Boutkova</i>	
Prioritizing Requirements: An Experiment to Test the Perceived Reliability, Usability and Time Consumption of Bubblesort and the Analytical Hierarchy Process	37
<i>Geurt Johan Van Tuijl, Wouter Leenen, Zhengru Shen, Inge Van De Weerd and Sjaak Brinkkemper</i>	
Speed Creation Session; A new way to increase the productivity of experts in projects and assure quality requirements	49
<i>Matthias M. D. Pohle, Sven Krause and Andreas Rusnjak</i>	

Faster from Requirements Documents to System Models: Interactive Semi-Automatic Translation

Leonid Kof, Birgit Penzenstadler

Fakultät für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748, Garching bei München, Germany
kof|penzenst@informatik.tu-muenchen.de

Abstract. [Motivation:] Natural language is the main presentation means in industrial requirements documents. This leads to the fact that requirements documents are often incomplete and inconsistent. Furthermore, if system models are constructed on the basis of these documents, explicit traces between the documents and the models are mostly missing.

[Problem:] Despite the fact that documents are mostly written in natural language, natural language processing (NLP) is barely used in industrial requirements engineering. Trade-offs of the existing NLP approaches hamper their broad usage: existing approaches either introduce a restricted language and, correspondingly, are able to process solely this restricted language, or, in the case of a non-restricted language, they cannot adapt to different writing styles, often co-existent in a single requirements document.

[Principal ideas:] Efficiency of system model construction and tracing can be improved by automated translation. Valuable side effects are early verification of requirements documents as well as tracing between the textual document and the constructed model.

[Contribution:] The presented paper shows how an NLP approach can be integrated in a CASE tool. The tool learns on the fly which grammatical construction represents which model element and enables text-to-model translation without restricting the allowed document language. The applicability of the tool in several case studies points towards improved efficiency for requirements analysis and design by using model generation from natural language requirements documents.

Keywords: requirements analysis, system modeling, natural language processing, automated generation, training

1 Requirements Documents Suffer from Missing Information

At the beginning of every software project, some kind of requirements document is usually written. The majority of these documents are written in natural language, as the survey by Mich et al. shows [1]. This results in the fact that the

requirements documents are imprecise, incomplete, and inconsistent, because precision, completeness and consistency are extremely difficult to achieve using mere natural language as the main presentation means.

It is one of the goals of requirements analysis, to find and to correct the deficiencies of requirements documents. A practical way to detect errors in requirements documents is to convert informal specifications to system models. In this case, errors in documents would lead to inconsistencies or omissions in models, and, due to more formal nature of models, inconsistencies and omissions are easier to detect in models than in textual documents.

Despite many years of research in natural language processing (NLP), the transition from textual documents to system models is still performed manually in industrial praxis. This is due to the fact that existing NLP approaches aiming at text-to-model translation introduce restricted languages and can process solely texts written according to the restrictions. So far, restricted languages are barely accepted in industry. Another class of NLP approaches do not assume any language restrictions. However, these approaches do not extract complete models from the text, they mostly extract just lists of terms used in the document. Furthermore, they cannot adapt to different writing styles, often co-existent in a single requirements document.

Contribution The paper shows how an NLP approach can be integrated in a CASE tool and enriched with machine learning techniques, so that the integrated system provides text-to-model translation without constraining the allowed natural language.

2 Interactive Text-to-Model Translation

This section presents the tool concepts, the training data collection, the semi-automatic system model extraction procedure, and the evaluation of the tool.

2.1 Motivation: Statistical Machine Translation

The text-to-model translator presented in this paper is motivated by statistical machine translation and other statistical techniques for natural language processing. The basic idea of statistical machine translation [2] is to subdivide the translation process into two phases, namely training and actual translation.

The same basic idea is applied for statistical part-of-speech (POS) tagging [3] and parsing [4]: First, the tagger/parser takes a set of manually tagged/parsed sentences, as for example the Penn Treebank (<http://www.cis.upenn.edu/~treebank/>), and gathers statistics on POS tags or parse trees constituents. Then, given the statistical data and an input sentence, the tagger/parser produces the most probable sequence of POS tags, resp. the most probable parse tree for the given sentence.

We decided to integrate the collection of the training data, the training, and the actual text-to-model translation in one tool as there is no sufficiently

sized training database available yet. The integration allows to gather document-specific training data, and to train the translator on these document-specific data. Thus, in our tool, the text-to-model translation consists of manual modeling and automatic extraction of model elements. Manual modeling is used to gather the training data, whereas automatic extraction analyzes the training data and proposes new model elements.

The details of the training data collection, training, and actual translation are presented below: Section 2.2 presents the training data collection, and Sections 2.3 and 2.4 present the training and actual text-to-model translation for different types of model elements.

2.2 Training Data Collection: Manual Modeling

The goal of manual modeling, considered as a part of text-to-model translation, is to gather training data for the subsequent automatic translation. From the point of view of user interaction, the tool distinguishes two types of model elements:

- entities, characterized solely by their names, and
- relations, characterized both by their names and the entities involved in the relation.

For example, a state of an automaton is an entity, as it is characterized solely by its name, but a state transition is a relation, as it is characterized by its name, and two states involved in the transition.

To create a model element, the tool user has to mark a word sequence in the text field showing the requirements document, and then to select the model element type from the context menu. The marked word sequence becomes the name of the newly created model element.

When creating a new model element, the tool creates an explicit trace between the model element and the previously marked word sequence. Every trace specifies the sentence, the exact position of the word sequence used to create the model element name, and the model element itself. These traces are used later as training data items for the automatic text-to-model translation.

2.3 Automatic Extraction of Entities

From the point of view of the tool user, extraction of entities looks similar to the manual creation of model elements: the user selects a document section, and then chooses from the context menu, which model element type should be extracted. Then, the tool proposes the names of potential new model elements to the user. The user selects, which names should be really used to create new model elements.

Internally, the tool keeps track of the user decision and augments the training data: for every model element name selected by the user, the tool creates a trace from the name occurrence in the text to the newly created model element, and makes the created trace to a positive example (new training data item). Every

not selected name becomes a negative example and is not presented to the user again. The tool does not distinguish between manually created and automatically extracted model elements and puts them in the same sets of positive/negative examples, as shown in Figure 1. The fact that the tool does not distinguish between manually created and automatically extracted model elements implies an important advantage: the tool does not force the user to create all the training data first, and then to perform automatic extraction only. Instead, the tool allows for arbitrary interleaving of manual modeling and automatic extraction steps.

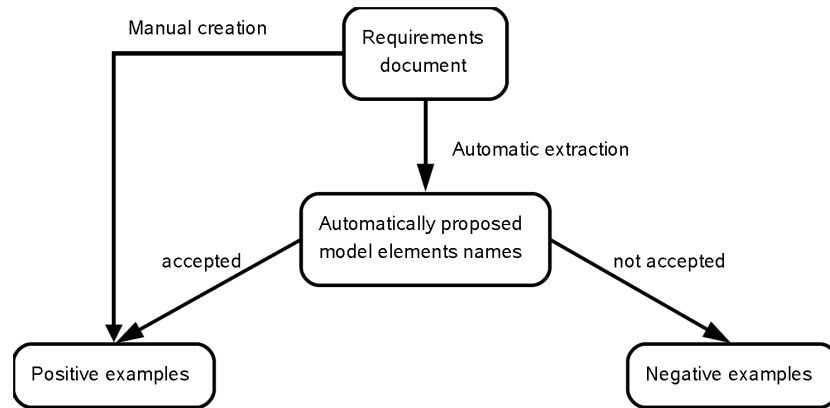


Fig. 1. Training data management

For automatic extraction of model element names, the tool performs the following steps:

1. **Trace detection:** turn every name occurrence into a training data item.
2. **Sentence properties analysis:** gather statistics on types of sentences with traces.
3. **Heuristics initialization:** decide on most suitable heuristics for trace.
4. **Heuristics application:** provide set of term extractions, present to user.

The extraction steps are presented below in detail.

Trace detection Before starting the extraction of model elements names, the user has to select a document section. The rationale for this design decision is the fact that industrial requirements documents are often written by different authors, using different writing styles. Both the learning procedures and the extraction of model elements names are more efficient when applied to a homogeneously written document section.

When the user selects the document section and the model element type to be extracted, the tool searches for the traces of the selected model element type in the selected document section. The rationale is that the names of different

model element types can be linguistically different, so the learning procedure is always applied only to the traces of the model element type selected by the user.

Training data gathered in this way are used in the subsequent learning and extraction steps.

Sentence properties analysis In order to handle compound sentences, the tool uses the information provided by the parser and splits every compound sentence into elementary parts (segments), cf. [6]. A segment is defined as a sentence part containing at most one verb phrase. Then, the tool gathers statistics regarding the properties of sentence segments containing traces.

The tool gathers statistics on three properties of the sentence segments containing traces: (1) segment contains “if” or “when”, (2) segment contains passive voice, and (3) main verb of the segment. The first two properties are binary: the tool counts the positive/negative examples contained in the sentence segments having the analyzed property. If the difference between the positive and negative examples exceeds the predefined threshold, the tool activates the corresponding property filter. Activated property filters are used for the extraction of new model element names. For the last property (main verb), the tool picks the verb used in the most sentence segments containing positive examples. Verbs are identified by means of part-of-speech tags provided by an external tagger.

As a first approximation, the tool extracts new model elements names from sentence segments having the same properties as the sentence segments containing the names of the previously created model elements.

Heuristics initialization In order to extract model elements names, the tool provides a set of heuristics. Most extraction heuristics are based either on the part-of-speech tags or simply on the sentence structure:

- Heuristics based on the work by Hearst [7]: The tool looks for two special sentence structures, namely “X, such as A, B, and C”, and “X ⟨some-text⟩: A, B, and C”. In both cases, A, B, and C can be extracted.
- The tool can extract verbs with their prepositions.
- The tool can extract whole sentence segments.
- Named entity recognition (NER): A named entity, in the context of the presented method, consists of a keyword, followed by a sequence of nouns or adjectives (“mode emergency stop”, “mode initialization”,...) The keyword can also be the last word of the named entity (“emergency stop mode”, “initialization mode”). The keyword for named entity recognition is derived from the training data.

In order to determine, which heuristics should be used for the actual extraction of new model elements names, the tool determines the motivating heuristic for every trace occurring in the training data. To determine the motivating heuristic for a given trace, the tool sequentially applies all available heuristics to the traced sentence, and picks the heuristic whose extraction result is most close to the traced word sequence. For example, “normal mode” in Sentence 1

normal mode

1. the normal mode is the standard operating mode in which the program tries to maintain the water level in the steam-boiler between n1 and n2 with all physical units operating correctly.
2. as soon as the water level is below n1 or above n2 the level can be adjusted by the program by switching the pumps on or off.
3. as soon as the program recognizes a failure of the water level measuring unit it goes into rescue mode.
4. failure of any other physical unit puts the program into degraded mode.
5. if the water level is risking to reach one of the limit values m1 or m2 the program enters the mode emergency stop.
6. this risk is evaluated on the basis of a maximal behaviour of the physical units.
7. a transmission failure puts the program into emergency stop mode.

Table 1. Steam Boiler Specification [5], excerpt

from Table 1 can be extracted with two heuristics: “NER” with keyword “mode” and “sentence subject”. In order to determine the motivating heuristic in such a case where the same word sequence can be extracted with several heuristics, the tool applies a priority list and picks the heuristic with the highest priority. The priority list results from our previous work [8]: the better results a heuristic provided in our previous studies, the higher its priority:

$$\begin{aligned} \text{priority}(NER) &> \text{priority}(verb + preposition) > \\ \text{priority}(Hearst) &> \text{priority}(subject/object) > \\ \text{priority}(before\ subject/after\ object) &> \\ \text{priority}(whole\ sentence\ segment) & \end{aligned}$$

Every heuristic that motivates some word sequence contained in the training data, and, in the case of several heuristics motivating the same word sequence, has higher priority than its competitors, is used to extract new model elements names.

Heuristics application To extract new model element names, every heuristic motivating a certain training data item is applied to the whole document section selected by the user. In order not to annoy the user with already accepted or already rejected model elements names, the tool purges the positive and negative examples from the set of the extracted model elements names (cf. Figure 1). In order not to overwhelm the user with too many new model elements names, the tool filters the sentences that are used for the extraction: First, the tool extracts new model element names exclusively from sentences with the same properties as determined above (see paragraph “Sentence properties analysis”).

If the number of newly extracted model elements lies below the predefined threshold, the tool successively attenuates the sentence property filter. Attenuation of binary properties (“if”-containment and passive voice) means that the

particular property filter is simply turned off. Attenuation of the verb property takes place in two steps: for the fully activated property filter, the tool picks the sentence segments containing the particular verb. After the first attenuation step, the tool picks the sentence segments containing some verb. After the second attenuation step, the property filter is turned off. As soon as the number of newly extracted model elements rises above the threshold or all the property filters are turned off, the attenuation and the extraction procedure terminate.

The tool presents the extraction results in a dialog (see Figure 2). In order to make the extraction transparent, the tool groups the extracted model elements names by the heuristic used for the extraction, and presents the heuristic and the used property filters together with the extracted model elements names. Then, the user has to select, which of the proposed model elements become real model elements. This feedback procedure augments both the model and the training data, as shown in Figure 1.

Currently, the extraction tool supports three types of entities: components, states, and MSC (Message Sequence Charts) actors.

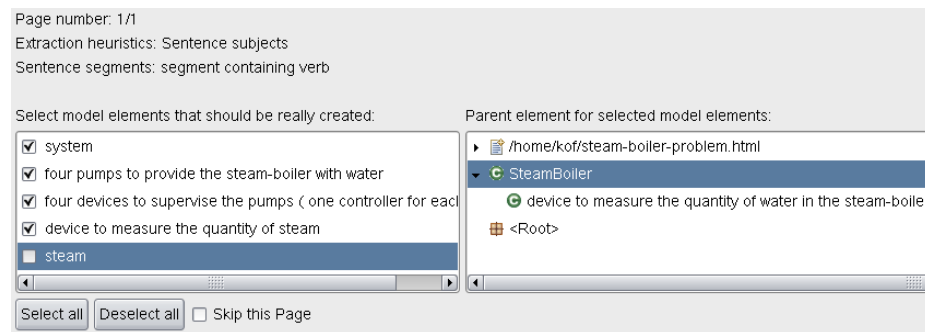


Fig. 2. Automatic extraction of model elements, selection of new model elements to be created and a container

2.4 Automatic Extraction of Relations

In order to extract relations (for example, state transitions), it is necessary not only to extract the name of the relation, but also to determine the entities involved in the relation. (E.g., in the case of state transitions, it is necessary to determine the involved states.) For proper identification of entities involved in a relation, a discourse model is necessary: For example, Sentence 5 from Table 1 states that the system switches from “normal mode” to “emergency stop mode” under certain circumstances. Although “normal mode” does not occur in Sentence 5, it is perfectly obvious for the human reader that “normal mode” is the source state of the transition specified in Sentence 5.

In order to find the entities involved in a relation, the tool uses a simple discourse model, similar to the model that was proven to be useful in our previous work [9].

In order to extract relations, the tool extracts relation names first, using the procedures presented in Section 2.3. Then, for every relation name that was accepted by the user, the tool looks for entities potentially involved in the relation: It follows text paths obtained from the training data and searches for the names of entities already existing in the model. For example, if Sentence 5 from Table 1 is the only training data item for state transitions, the tool will always look for the source state in the header of the section containing the name of the extracted state transition, and for the target state in the same sentence as the name of the extracted state transition.

The found entities, potentially involved in the relation, are presented to the user. The user selects, which entities are really involved in the relation, and after that the tool creates the relation, using the extracted relation name and the selected entities.

Currently, the extraction tool supports four types of relations: channels, state transitions, MSC messages, and MSC assertions (no explicitly involved entities).

3 Evaluation

The tool was evaluated on four specifications with respect to applicability: Steam Boiler [5], Autopilot [10], Bay Area Rapid Transit (BART) [11], and Instrument Cluster [12]. The goal of the evaluation was to investigate if the extraction results keep up with the expectations of the tool author. Due to possible evaluator’s bias, no further evaluation was performed. The remainder of this section presents the evaluation results for the four specifications used for the evaluation.

Steam Boiler. This specification describes the steam boiler itself and states the requirements to the control program for the steam boiler. The goal of the control program is to maintain the water level, even if certain equipment fails, between predefined marks, in order to prevent damage of the steam boiler. To construct the model, “steam boiler” was manually marked as a component. The tool managed to extract the remaining components automatically. Then, “initialization” (mode) was manually marked as a program state, and the tool extracted all other operating modes. To extract state transitions, one “if” sentence, similar to the Sentence 5 from Table 1, was used as a training data item. This enabled the tool to correctly extract all state transitions encoded in “if”-sentences. In the second extraction step, the tool attenuated the sentence property filter and was able to extract state transitions like “as soon as ...” (Sentence 3 from Table 1 and similar), as well as “failure of ... leads to ...” (Sentence 4 from Table 1 and similar). Finally, Sentence 7 from Table 1 was manually marked as a state transition. The tool extracted similar sentences as state transitions, but also produced some noise, that had to be filtered out manually in dialogs like the dialog shown in Figure 2.

Autopilot. The autopilot case study is extremely short (just 2 pages of requirements) and describes the user interface (buttons and displays) and the operating modes of an autopilot system. Two operating modes had to be manually marked as states, then the tool extracted the remaining modes. Similarly, one of the displays was marked as a component, and the tool extracted the other displays as components. As the specification does not explicitly specify state transitions or connections between components, no further extraction was performed.

BART. The BART (Bay Area Rapid Transit) specification describes the train control system itself and safety requirements that must be satisfied by the control system. To model the BART system, “non-vital station computer” was manually marked as a component, and the tool extracted other controllers (“vital station computer”, “station computer”) as components. Furthermore, as the specification contains precise information on different phases of the braking process, these informations were modeled too. Here, we used mock up modeling: we manually marked the name of one of the braking phases as a component, and the tool was able to extract the other breaking phases. The resulting model does not really make sense, but it allowed us to evaluate the linguistic part of the tool. As the specification does not explicitly specify state transitions or connections between components, no further extraction was performed.

Instrument cluster. The Instrument Cluster specification describes the optical design of the instrument cluster as a part of the car dashboard, its hardware, and, most importantly, its behavior. Behavior specification consists of 10 use cases, with several scenarios for every use case. To translate scenarios to message sequence charts, “driver” and “car” were manually marked as components, then the tool was able to extract further components involved in scenarios (“instrument cluster”, “ignition”, “motor”, . . .). After the extraction of components, “switches on”, taken from the sentence “The driver switches on the car”, was manually marked as an MSC message. Then, the tool automatically extracted further messages (verbs from different sentences). To summarize, in all case studies, the tool was able to keep up with the expectations of the tool author.

4 Related Work

Work related to the presented paper can be subdivided in two areas: work on text-based modeling and work on natural language processing (NLP) in requirements engineering.

4.1 Text-Based Modeling

Saeki et al. [13], Overmyer et al. [14], and Ermagan et al. [15] introduced tools providing modeling approaches related to the approach presented in this paper. The approach by Saeki et al. allows the user to mark words in the requirements documents, and then assign them to a word type. Then, the approach maps nouns to classes (object-orientated), and verbs to operations, but can handle

just four predefined verb classes and is not able to learn on the fly, as the approach presented in this paper. Overmyer et al. developed a tool allowing the user to mark words or sequences and map them to classes, roles, and operations. They do not assume that the verb must fall into one of the predefined categories. Nevertheless, the tool by Overmyer et al. cannot learn either. Ermagan et al. developed the tool SODA, allowing to link textual use cases to behavior models. Such links are similar to traces introduced in the presented paper, but SODA sticks to manual modeling.

4.2 Natural Language Processing in Requirements Engineering

Natural language processing is applied in assessment of document quality, identification and classification of application specific concepts, and analysis of system behavior.

Assessment of document quality. Rupp [16], Fabbrini et al. [17], Kamsties et al. [18], and Chantree et al. [19] define writing guidelines and measure their satisfaction. Their aim is to detect poor phrasing and to improve it; they do not target at system modeling, as our approach.

Identification of application specific concepts. Goldin and Berry [20], Abbott [21], or Sawyer et al. [22] analyze the requirements documents, extract application specific concepts, and provide an initial static model of the application domain. However, these approaches cannot learn on the fly which concepts should be extracted, and they do not perform any behavior modeling.

Analyzing system behavior. Vadera and Meziane [23], Gervasi and Zowghi [24], Breaux et al. [25, 26], and Avrunin et al. [27] translate requirements documents to executable models by analyzing linguistic patterns. Vadera and Meziane propose a procedure to translate certain linguistic patterns into first order logic and then to the specification language VDM, but they do not provide automation for this procedure. Gervasi and Zowghi go further and introduce a restricted language, a subset of English. They automatically translate textual requirements written in this restricted language to first order logic. Similarly, Breaux et al. introduce a restricted language and translate this language to description logic. The approach by Avrunin et al. translates natural language to temporal logic.

Our work goes further than the above approaches, as we do not assume or enforce language restrictions. Treatment of non-restricted language is possible due to the ability of the presented approach to learn on the fly.

To summarize, to the best of our knowledge, there is no approach to documents analysis, yet, able to translate model descriptions written in non-restricted natural language to models themselves, and to learn on the fly which phrases should be translated to which part of the model.

5 Discussion and Conclusion

Both requirements engineering and system modeling are non-trivial tasks and the presented approach does not claim to solve all their problems. However, it provides important links between textual requirements and modeling:

- **Efficient analysis and early construction:** Given a textual requirements document, it helps to explore the document and to construct a system model.
- **Automated tracing:** When constructing the model, it maintains explicit traces between the model and the textual document.
- **Early verification of requirements documents:** If the model description in the document is incomplete, it makes this incompleteness apparent, by creating an incomplete system model.

In this way, the presented approach closes the gap between textual specifications and models in an efficient way to support lean and fast requirements engineering.

Benefits. The main benefit of the integration of an NLP approach is not the extracted model itself, but the extraction procedure: by extracting both model elements and relations between them, NLP would contribute to thorough exploration and thereby early verification of the requirements document. These corrections can also result in the necessity to change the specification text. Thus, the CASE tool with integrated linguistic techniques provides means for tracing model elements back to the specification text. Although a specific tool was used to demonstrate the main idea (AutoFOCUS <http://af3.in.tum.de/>), the linguistic part of the approach remains tool-independent.

Limitations. Complete translation does not mean that the resulting model can be directly used for system simulation or code generation. Both due to inconsistencies and omissions in the requirements document, and due to imperfection of the linguistic tools, the resulting model will most probably need corrections, before it can be used for system simulation or code generation. Nevertheless, the partial automation and the integration of requirements analysis and early design improve the efficiency of requirements engineering processes.

Outlook. The linguistic part of the approach presented in this paper is generic: it does not depend on the CASE tool, nor on the types of model elements or relations that should be extracted. This implies that the presented approach to text-to-model translation can be integrated into every CASE tool, thus allowing us to translate a requirements document to the model most appropriate for the particular application domain. This, in turn, ensures industrial applicability of the presented approach.

References

1. Mich et al.: Market research on requirements analysis using linguistic tools. *Requirements Engineering* **9** (2004)
2. Koehn, P.: *Statistical Machine Translation*. Cambridge University Press (2010)
3. Curran et al.: Multi-tagging for lexicalized-grammar parsing. In: 21st International Conference on Computational Linguistics. (2006)
4. Clark & Curran: Parsing the WSJ using CCG and log-linear models. In: *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. (2004)
5. Abrial et al.: The steam boiler case study: Competition of formal program specification and development methods. In: *Formal Methods for Industrial Applications*. (1996)

6. Kof, L.: Treatment of Passive Voice and Conjunctions in Use Case Documents. In: Application of Natural Language to Information Systems. (2007)
7. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Proceedings of the 14th conference on Computational linguistics. (1992)
8. Kof, L.: Requirements Analysis: Concept Extraction and Translation of Textual Specifications to Executable Models. In: Application of Natural Language to Information Systems. (2009)
9. Kof, L.: Translation of Textual Specifications to Automata by Means of Discourse Context Modeling. In: Requirements Engineering: Foundation for Software Quality, 15th International Working Conference. (2009)
10. Butler, R.W.: An introduction to requirements capture using PVS: Specification of a simple autopilot. Technical report, NASA Langley Research Center (1996)
11. Winter et al.: The BART case study. In: Formal Methods for Embedded Distributed Systems. (2004)
12. Buhr et al.: DaimlerChrysler demonstrator: System specification instrument cluster. Project Empress ITEA (2004)
13. Saeki et al.: Software development process from natural language specification. In: Proc. of the 11th Intl. Conf. on Software Engineering. (1989)
14. Overmyer et al.: Conceptual modeling through linguistic analysis using LIDA. In: Proc. of the 23rd Intl. Conf. on Software Engineering. (2001)
15. Ermagan et al.: Towards Tool Support for Service-Oriented Development of Embedded Automotive Systems. In: Proceedings of the Dagstuhl Workshop on Model-Based Development of Embedded Systems (MBEES'07). (2007)
16. Rupp, C.: Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis. Hanser-Verlag (2002)
17. Fabbrini et al.: The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In: 26th Annual NASA Goddard Software Engineering Workshop. (2001) http://fmt.isti.cnr.it/WEBPAPER/fabbrini_nlrquality.pdf, accessed 08.02.2010.
18. Kamsties et al.: Detecting ambiguities in requirements documents using inspections. In: Workshop on Inspections in Software Engineering. (2001)
19. Chantree et al.: Identifying nocuous ambiguities in natural language requirements. In: Proc. of the 14th IEEE Intl. Requirements Engineering Conference. (2006)
20. Goldin & Berry: AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Eng.* **4** (1997)
21. Abbott, R.J.: Program design by informal English descriptions. *Communications of the ACM* **26** (1983) 882–894
22. Sawyer et al.: Shallow knowledge as an aid to deep understanding in early phase requirements engineering. *IEEE Trans. Softw. Eng.* **31** (2005)
23. Vadera, S., Meziane, F.: From English to formal specifications. *The Computer Journal* **37** (1994) 753–763
24. Gervasi, V., Zowghi, D.: Reasoning about inconsistencies in natural language requirements. *ACM Trans. Softw. Eng. Methodol.* **14** (2005) 277–330
25. Kiyavitskaya et al.: Automating the extraction of rights and obligations for regulatory compliance. In LNCS, ed.: ER. Volume 5232. (2008)
26. Breaux et al.: Semantic parameterization: A process for modeling domain descriptions. *ACM Trans. Softw. Eng. Methodol.* **18** (2008)
27. Smith et al.: PROPEL: an approach supporting property elucidation. In: Proceedings of the 24th International Conference on Software Engineering. (2002)

Efficiency in Requirements Engineering through Feed-forward

Sharmin Ahmed, Muhammad Iftekher Chowdhury,
Remo Ferrari and Nazim H. Madhavji

Department of Computer Science, University of Western Ontario,
London, Canada, N6A 5B7
{sharmin.ahmd, iftekher.chowdhury, ferrari.remo, madhavji}@gmail.com

Abstract. There are different ways in which the efficiency improvement of a requirements engineering (RE) process can be addressed. Examples are: automation through tool support, reducing peripheral documentation, reducing the number and types of artefacts processed, detailing requirements only when needed, etc. While such approaches may at first seem to increase the efficiency, there may well be negative long-term consequences, for example, in terms of increased backtracking and rework, lower quality of downstream artefacts and the eventual product, delayed delivery of the system, etc. Such consequences counteract the initial gains in RE. In this vision-category paper, we present a new idea, that of a feed-forward (FF) centric RE process. FF is an “anticipatory activity” of passing information from one location in the RE process to another, where the recipient of the information could possibly benefit by making use of it “ahead of time”. This could result in savings of time; avoidance of unnecessary activities; execution of necessary, but as yet not considered, activities; etc., all leading to improved RE decisions. While FF is known in other domains such as systems engineering and management, to our knowledge, it has not been scientifically investigated or formulated as a vision in RE. In this paper, we discuss the concept of FF in terms of the five Ws and one H (i.e., why, what, where, when, who, and how) and also present example requirements for enabling FF in an RE process.

Keywords: Requirements Engineering, Software process improvement, Feed-forward, Software team communication, agile processes.

1 Introduction

There are different ways in which the efficiency improvement of a requirements engineering (RE) process can be addressed. Example approaches include: automation through tool support [16], reducing peripheral documentation [6], reducing the number and types of artefacts processed [6], detailing requirements only when needed [6], etc. While such approaches may at first seem to increase RE efficiency, there may well be negative long-term consequences. For example, a study involving 16 organisations practising agile approaches shows that the quality of downstream artefacts is reduced if less emphasis is placed on security and performance

requirements early in the RE process, ultimately resulting in increased backtracking and rework [4]. Similarly, research described in [9] shows that reduced front-end documentation leads to information loss which, in turn, can have a negative impact on the product and process quality over time. Furthermore, though locally focused tools (e.g., RE-only tools) may lead to improved local artefacts and processes, they can also lead to fragmentation across different life-cycle processes (e.g., architecting) because of the lack of tool integration across such processes, which can limit developer capabilities and productivity [15]. Such consequences counteract the initial efficiency gains in RE.

In this vision-category paper, we present an orthogonal idea of a “feed-forward” (FF) centric process for improving efficiency in RE. FF is defined in non-software literature as: “... some output of an earlier step is fed into a step occurring down the line” [7]. For software projects, we interpret FF as: “an anticipatory activity of passing information from one location in the process to another where the information could possibly be used ahead of time”. Note here that “ahead of time” is quite compatible with the idea of “down the line” in the definition and has a direct impact on process efficiency. Also, the FF information need not be intra-process, meaning that the source of information could lie outside the affected process [5]. Contrast this with the “normal flow” of the development process; information forwarding is not anticipatory and is not passed ahead of time. Feed forwarding information under favourable conditions can aid recipients to get on with their tasks and decision-making, possibly even altering the course of action they would take because of the FF information they have received.

The notion of FF is already being used in other domains such as systems engineering [13] and management [10]. However, in RE and software engineering (SE) practice, the concept of FF is not engrained in the process (contrast this with the concept of “feedback”); there are only anecdotal references to it [1], [8], [15].

Our vision of FF in RE is particularly motivated from our observations from a previous empirical study [15]. We postulated that if architects had access to critical RE information prior to the validation of the full set of requirements, then this FF information could trigger early groundwork for specific architectural enhancements, while other requirements were still being elicited. Such agile use of FF information has potentially significant benefits for RE process efficiency, in particular: effort minimization; early highlighting of implementation risks; minimising effort expended; ensuring a tighter coordination between varying project roles; etc.

Note that these benefits and usage of FF complement those from other RE efficiency approaches (for example, process improvement [17], automation [16], etc.). FF is not meant to supplant these existing approaches but can be used in conjunction to increase RE efficiency.

In the next section, we describe related work, following which, in Section 3, we discuss the concept of FF in terms of the five Ws and one H (i.e., why, what, where, when, who, and how). In Section 4, we give example requirements for enabling FF in an RE process. Finally, Section 5 summarises the paper and mentions our on-going work.

2 Background

In this section, we first overview examples of FF from other disciplines (control engineering, artificial neural networks, and management) before describing examples of FF in SE.

A well-known application of FF is in the domain control engineering [13], where FF in control systems are “open loop” systems where corrective action is taken *before* measuring and acting upon the output of the system. This is in contrast to “closed loop” systems where corrective action (feedback) is taken *after* an error is detected in the output and is reported back to the source. The thermostat of an house is an example [5] of a closed loop feedback controller. This controller relies on measuring the controlled variable (the temperature of the house), and then adjusting the output (to turn on/off the heater). However, feedback control usually results in periods of time where the controlled variable is not at the desired setpoint (i.e., the room temperature is below or above the set temperature on the thermostat). Thus, if the door of the house were opened on a cold day, the house would cool down. After it fell below the setpoint, the heater would turn on, but there would be a period when the house is colder than desired. FF control can avoid the slowness of feedback control. With FF control, the disturbances (i.e., events other than inside-house temperature) are measured and accounted for before the feedback controlled affects the system. Thus, a FF system may measure the fact that the door is opened and automatically switches on the heater before the house gets too cold.

Other examples of FF are in artificial neural networks [11], where pre-determined information is fed forward into the system and the parameters of the network are adjusted to minimise deviations from the expected output. In [10], Goldsmith discusses how a ten-minute exercise on FF practised by business professionals is preferable to feedback for day-to-day interactions. The reason is that feedback is often taken personally as a negative but FF does not suffer this fate.

In SE, we also observe examples of FF. In [1], Agresti mentions how anticipated staff-turnover in the forthcoming weeks - a piece of human-resource information that may not be known in a given software project - can be fed forward to the appropriate project manager(s) so that it can be incorporated into the time-estimation models to recalculate the project estimates and to assess risks due to staff turnover. In [8], Fricker et al., propose a goal-oriented systems model for the communication of requirements from product management to development teams. One paradigm described in this model explicitly mentions the use of FF, where the product manager uses their knowledge of the development teams' expertise to anticipate the project output and accordingly adjust the requirements to be elicited. In [12], Lehman et al., mentions that software project and process issues (such as specification changes, unexpected conditions, performance problems) can be overcome through proper usage of a mixture of feedback and feed-forward in software process.

While FF is mentioned in SE [1], [12], [15] and RE [8], no scientific research on FF has been conducted to our knowledge. This paper is a first step to fill this gap.

3 The Concept of Feed-forward in the RE context

In this section, we analyse the concept of FF, in the context of RE, in terms of the five Ws and one H¹ (i.e., who, when, where, why, what, and how) to gain a better understanding of the underlying issues.

Who and When: Any project stakeholder (e.g., product manager, release manager, RE analyst, etc.) within a RE process can potentially feed information forward to another stakeholder. A key precondition for FF is that it must occur *ahead of time* of a given event in the process. For example, analysts can feed forward critical requirements information to the software architect during elicitation *prior* to requirements validation so that architect can take preparatory early action.

Where: The idea of FF mentioned in [15] is an example of “inter-process” FF, where information is fed forward, in this case, from the RE process to the software architecting process. FF can also occur “intra-process”. For example, in a large distributed development effort, a sub-team refining the high-level requirements for their components could influence the (say, prioritising) decisions being made by another sub-team.

Why: In the Introduction section, we described the overall motivation and rationale behind incorporating FF in the RE process. Here, we list several additional reasons for practising FF in RE. This rationale is based on general management literature [10] but it can also apply to RE.

- FF tends to be faster and more efficient than feedback.
- We cannot change the past but we can change the future.
- It can be more productive to help people to be right, than to prove that they were wrong.
- FF is not taken as personally as feedback.
- Unlike FF, feedback can reinforce personal stereotyping and negative self-fulfilling prophecies.
- People do not like giving or receiving negative feedback. On the other hand, FF is always positive because it concerns the future.

What: Any type of information can potentially be fed forward (example: requirements, costs data, implementation difficulty and risk information, etc.).

How: Information can be fed forward using any possible means of communication used in the RE process, such as e-mail, face-to-face meetings, instant messaging, telephone, video-conferencing, memos and other forms of documentation. Teams or individuals may be co-located or distributed, and based on the distribution the format of the communications may vary. Also, FF information can be communicated indirectly through other people in the organisation, or more commonly referred to in the literature as information or knowledge brokers [3] [14].

¹ A well-known formula in journalism and research ([2], p.8-10) to cover a topic comprehensively.

4 Example Requirements

While the idea of FF is appealing, omnipresent evidence suggests that, unlike feedback, it is not a “needs-driven” activity. In the case of feedback, for instance, we know readily that the quality of the emerging requirements artefacts and of the eventual solution are going to suffer without integrating feedback into the process. Even a low amount of feedback is likely to yield tangible gains in quality. On the other hand, in the case of FF, the benefits are not as clear to everyone, though there is hope. Also, effective FF requires an elevated level of thinking and judgment from the stakeholders involved. In terms of maturity of the RE process, one can argue that “feedback” is necessary in low maturity processes; whereas, existence of cost-effective and fruitful FF in a process would suggest an incredibly high maturity of the process.

Here, we list several example requirements to enable FF in an organisation’s RE process:

- For FF to be cost-effective in a project, individual stakeholders possessing FF information need to be familiar with other stakeholders’ tasks and activities that may benefit from the FF information.
- Stakeholders need to be trained on the concept of FF and on the conditions when FF should (or could) be, or not be, used.
- Organisational motivation (e.g., through recognition, rewards and incentives) and self-motivation are key factors in institutionalising successful FF in an organisation.
- There is a need to make the cost-benefit ratio of FF, from release to release, visible to the stakeholders (a form of feedback!) so as to make improvements with respect to FF.
- FF requires a high degree of organisational and project maturity.

These requirements implicitly suggest FF-related principles to be engrained in the fabric of the organisation. We do not, for even one moment, assume that implementation of these requirements is trivial. However, prior to any attempt at implementing such requirements it is important that empirical studies are conducted to ascertain the cost-benefits of FF.

5 Summary and On-going Work

This short paper describes the idea of a feed-forward (FF) centric process with the goal of improving RE efficiency. FF is an “anticipatory activity” of passing information from one location in the RE process to another or to other SE processes, and vice versa, where the recipient of the information can benefit by making use of the information ahead of time. The paper discusses the concept of FF in terms of the five W’s and one H (i.e., why, what, where, when, who, and how) [2]. Also presented are several example requirements (in Section 4) for enabling FF in an organisation’s RE process. We do not consider successful implementation of such requirements to be trivial because it requires embracing the idea of FF at all levels in an organisation. However, prior to embarking upon organisational change, it is important to conduct

empirical studies to assess the costs-benefits of FF. As part of our ongoing work on FF, we are conducting preliminary case studies and an industrial survey. These will be reported when the studies are complete. We hope that the ideas presented in this paper can act as a stepping-stone towards increased awareness of FF in the field of RE, eventually leading to the adoption of FF practices in industrial-scale RE processes.

References

1. Agresti, W.W.: A Feedforward Capability to Improve Software Reestimation. In: Madhavji, N. H. et al. (eds.) *Software Evolution and Feedback*, pp. 443--458. John Wiley & Sons, West Sussex, UK (2006)
2. Blaikie, N.: *Designing Social Research: The Logic of Anticipation*. Polity Press, Cambridge, UK (2000)
3. Boden, A., Avram, G.: Bridging knowledge distribution - The role of knowledge brokers in distributed software development teams. In: *31st International Conference on Software Engineering (ICSE'09) - Workshop on Cooperative and Human Aspects on Software Engineering*, pp. 8--11. IEEE Computer Society, Washington, USA (2009)
4. Cao, L., Ramesh, B.: Agile Requirements Engineering Practices: An Empirical Study. *IEEE Software*. 25 (1), 60--67, IEEE Computer Society, CA, USA (2008)
5. Controller (control theory) - Wikipedia, the free encyclopedia, [http://en.wikipedia.org/wiki/Controller_\(control_theory\)](http://en.wikipedia.org/wiki/Controller_(control_theory))
6. Davis, A.: *Just Enough Requirements Management: Where Software Development Meets Marketing*. Dorset House Publication Co., Inc., New York, USA (2005)
7. Feed forward definition, <http://www.businessdictionary.com/definition/feed-forward.html>
8. Fricker, S., Gorschek, T., Glinz, M.: Goal-Oriented Requirements Communication in New Product Development. In: *Second International Workshop on Software Product Management*, pp. 27--34. IEEE Computer Society, CA, USA (2008)
9. George, B., Bohner, S. A., Prieto-Diaz, R.: Software information leaks: a complexity perspective. In: *9th International Conference on Engineering Complex Computer Systems (ICECCS'04)*, pp. 239--248. IEEE Computer Society, CA, USA (2004)
10. Goldsmith, M.: Try Feedforward Instead of Feedback. *Leader to Leader*. 25, 11--14, Leader to Leader Institute, NY, USA (2002)
11. Haykin, S.: *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, Englewood Cliffs, NJ, USA (1999)
12. Lehman, M. M., Ramil, J. F.: Software Evolution and Software Evolution Processes. *Annals of Software Engineering*. 14, 275--309, J. C. Baltzer AG, Science Publishers, Red Bank, NJ, USA (2002)
13. Leigh, J. R.: *Control Theory*. Institute of Engineering and Technology, UK (2004)
14. Marczak, S., Damian, D., Stege, U., Schroter, A.: Information Brokers in Requirement-Dependency Social Networks. In: *16th International Requirements Engineering Conference (RE'08)*, pp.53--62. IEEE Computer Society, CA, USA (2008)
15. Miller, J. A., Ferrari, R., Madhavji, N.: An Exploratory Study of Architectural Effects on Requirements Decisions. *Journal of System and Software*. 83, 2441--2455, Elsevier Science Inc., NY, USA (2009)
16. Requirements Tools, <http://www.volere.co.uk/tools.htm>
17. Sommerville, I., Ransom, J.: An empirical study of industrial requirements engineering process assessment and improvement. *ACM Transactions on Software Engineering Methodology (TOSEM)*. 14, 85--117, ACM, NY, USA (2005)

Towards Faster Application Engineering through Better Informed Elicitation – A Research Preview

Sebastian Adam

Fraunhofer IESE, Fraunhofer Platz 1, 67663 Kaiserslautern
sebastian.adam@iese.fraunhofer.de

Abstract. Application engineering (AE) is still a time-consuming and expensive task, and the benefits from using a software product line (SPL) approach are often fewer than expected. Among others, one important reason for this low efficiency is the non-systematic mapping of customer requirements, due to the fact that requirements engineers are often not sufficiently informed about SPL characteristics. In this research preview, we present our current and planned work regarding this problem. We discuss how we plan to develop an approach that systematically guides the tailoring of elicitation instruction for application engineering requirements engineering (AERE). In this regard, we also discuss expected improvements and how we plan to evaluate them.

1 Introduction

As a key concept for fast and efficient software development, software product lines (SPL) [1] have proven to be a promising strategy. Nevertheless, developing new systems based on an SPL during the application engineering (AE) phase is still time-consuming and expensive [2], and the benefits from using an SPL approach are often fewer than expected [3].

Among others, one important reason for this low efficiency is the non-systematic mapping of customer requirements [4], even though it has been recognized that the success of AE mainly depends on how requirements are treated. On the one hand, current application engineering requirements engineering (AERE) approaches foster the direct reuse of SPL requirements rather than the effective alignment of a customer's actual needs with the available SPL capabilities [5] [6]. However, customer-specific development based merely on picking predefined requirements from a repository is typically not sufficient, because such systems also have to reflect a significant number of individual requirements in order to allow a customer to stand out from the competition. This is especially the case in the information system (IS) domain, in which very specific business processes have to be mapped to IT. Therefore, many costly rework iterations are typically needed until a delivered system fulfills its expectations.

On the other hand, eliciting customer requirements from scratch without considering SPL characteristics early on is also not an appropriate option. Particularly since selecting an SPL implies a certain set of constraints, it becomes apparent that not all customer requirements can be realized as initially stated. Rather, trade-offs

between ideal requirements and rapid development benefits must be made. However, making this trade-off is a challenging task, because information about the realizability of requirements is often neither formalized nor available in the early requirements phase. As a result, it is therefore often up to the development team in the back office “to bridge the gap between requirements and implementations” [7]. This typically leads to either costly rework or renegotiations until the number of non-fitting requirements is reduced. Unfortunately, checking the fit in such an analytical manner and improving the requirements afterwards is still the state of the art. Approaches, that handle this challenge better, do not exist yet [4] [5]. Thus, the question *how a satisfactorily fit between customer requirements and SPL characteristics can be achieved much faster* is still an unsolved practical problem to be addressed.

In our previous work, we have identified the fact that *AE requirements engineers are often not informed sufficiently about the underlying SPL* as a reason for this problem. We therefore currently deal with the research question on *how an AE requirements engineer can be enabled to use knowledge about a SPL for better guiding the elicitation of customer requirements*. This includes the sub questions on how to extract SPL knowledge and on how to represent this knowledge appropriately.

As a general idea to answer these questions, we have proposed the idea to tailor AERE processes based on the characteristics of a given SPL, and to provide very precise and informative elicitation instructions in this regard (see [8] [9]). Our fine-grained research objectives (RO) arising from this aim include a

- (1) **clarification of the role of requirements in AE** in order to explain how AERE processes should be aligned with a given SPL,
- (2) **definition of an appropriate structure for AERE elicitation instructions** that enable requirements engineers to work more effectively,
- (3) **formalization of tailoring steps** in order to provide a precise procedure for a reproducible reflection of SPL knowledge in these elicitation instructions,

In this paper, we present our current work regarding the first objective (see section 2) and we give an outlook how this clarification may support our ongoing and future research regarding objectives 2 and 3 (see section 3). Thus, the main contribution of this paper is a first clarification which requirements should be in the focus of AERE and how this knowledge can be used to design an AERE tailoring approach.

2 Current Research and Results

An important prerequisite to elaborate a tailoring approach that reflect SPL knowledge in an AERE process is a clear understanding on how AERE processes are related with an underlying SPL (RO 1). Our research effort spent so far has therefore dealt with the clarification of the role and interrelationship of requirements in AE.

Basically, “*requirements express the needs and constraints placed on a product that contributes to the solution of some real world problem*” [11]. However, as there is still no universal definition of a requirement [10], several taxonomies have been proposed in the RE standard literature in order to clarify what a requirement could be. What all these taxonomies have in common is that they only address the content of a requirement (e.g., functional vs. non-functional) and not the role a requirement may have with regard to development or reuse. In many of today’s AE projects, however,

one can observe that the set of **elicited** customer requirements is only partially **fitting** a given SPL. Based on our own experience with RE in industry projects, we have therefore recently developed a novel taxonomy that distinguishes requirements from the viewpoint of their relevance and realizability in AE.

As fitting requirements, we basically define those elicited requirements that cover the SPL's **variable** requirements or the SPL's **implicitly anticipated** requirements (i.e., those that are in-scope but not explicitly modeled already), and those elicited requirements that are not in scope but **incidentally realizable** anyway. Fitting requirements have therefore all in common that they are (economically) **realizable** with a given SPL and also needed for its instantiation or customer-specific extension. In this regard, we call the entire and probably infinite super set of all variable, implicitly anticipated, or incidentally realizable requirements a SPL may realize, the set of **relevant** requirements for this SPL. However, another, typically not small, subset of elicited requirements is often neither needed nor desired. First, there are requirements that are **unnecessary** because the solutions they ask for are implemented by default anyway. Hence, these requirements just repeat **common** SPL requirements. Second, there are elicited requirements that are not (economically) realizable with the given SPL. These **problematic** requirements could only be fulfilled with very high costs and risks because significant changes have to be made either to common implementations or even to the entire architecture. Problematic requirements typically result from an insufficient consideration of existing constraints and thus, from giving customers too much freedom. Besides problematic requirements, **missing** requirements are another risk. In particular, when developers do not get all the information they need for making decisions, they have to make their own assumptions or re-elicitation. Missing requirements typically arise from the fact that no concrete instruction is given on how to elicit which requirements.

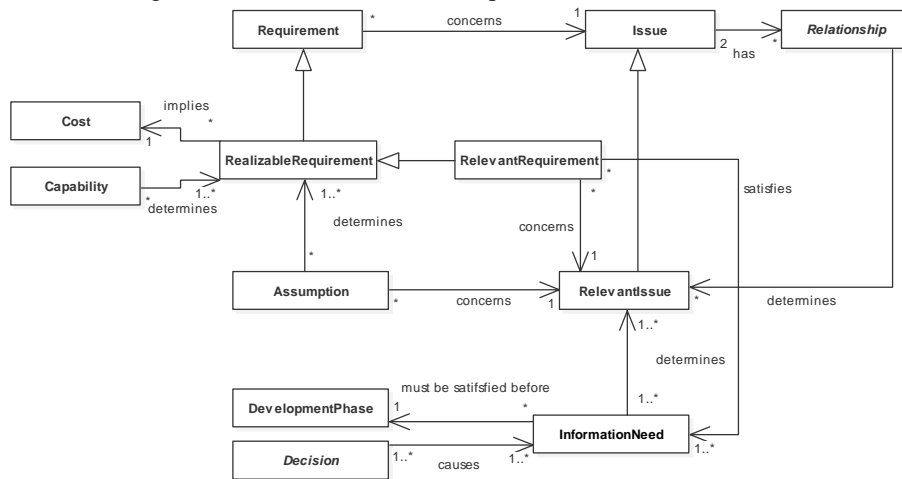


Fig. 1. The origin and relationships of relevant requirements

In order to better align AERE processes in a sense that all requirements are fitting and no important requirement is missing, we have currently developed a conceptual model with around 75 elements in ten views that clarifies how AERE processes

should be aligned with a given SPL. In this model, from which we only show two views in this paper (see Fig. 1 and Fig. 2), especially “information needs” and “assumptions” are essential elements to explain how knowledge about a SPL can be appropriately extracted.

Basically, a **requirement** is a piece of information concerning an issue. As **issue** we define a class of elements that are either part of a system or part of the system’s usage environment. Issues therefore cover all functional and non-functional system aspects (e.g., use cases), but also elements of the usage environment for which a system must be designed in order to provide appropriate support (e.g., users, work places, data, devices, etc.). As issues represent the real world, there may be **relationships** between them; for instance, between users and their workplaces.

However, as mentioned before, not all requirements are (economically) realizable with a given SPL. The important drivers for determining **realizable requirements** are the already existing, explicitly anticipated requirements, as well as the set of **assumptions** that have been made during the design of the product line architecture. Thus, assumptions are descriptive rather than enumerative specifications of requirements that are realizable by a SPL. For requirements concerning the issue “user interface”, for instance, an assumption could be that only web-based clients are (economically) realizable. Of course, when considering the SPL’s common requirements, it becomes evident that not all **realizable requirements** are also relevant for deriving systems from an SPL. To refine the aforementioned definition, a relevant requirement is thus a realizable requirement that is concerned with a **relevant issue**. In this context, an issue is called relevant if a requirement concerned with it is able to satisfy an **information need** in the AE’s development process, or, if an issue is needed for the elaboration of such requirements. For instance, when information about workplaces is necessary to implement an appropriate user interface, “workplace” would be a relevant issue. And, also the “user” via whom the “workplaces” are to be identified, would be relevant implicitly.

Thus, besides assumptions, information needs are another central concept in our conceptual model to determine which requirements are relevant and which are not. In order to identify these information needs, the **decisions** to be made during an AE’s development process must be analyzed, as each decision basically causes at least one information need. In the workplace example, the decisions on how to implement a user interface have caused the need to know what the workplaces look like. Thus, before a **development phase** dealing with the development of user interfaces can start, this information need must be satisfied through the provision of realizable requirements concerning the workplaces to be supported.

3 Open Issues and Outlook

It is evident that knowledge about both the development processes and the product line architecture must exist in order to determine relevant requirements through the notion of information needs and assumptions. Currently, we are finalizing our conceptual model that explains how all SPL-related aspects belong together. An open issue in this regard is whether our model is correct and sufficiently complete. To answer this question, we plan to validate the model through expert reviews soon.

Besides the conceptual model, we have recently started to develop a tailoring approach (addresses RO 3) that explains how assumptions and information needs can be systematically extracted from a SPL and then reflected into precise elicitation instructions. The purpose of these elicitation instructions is to prescribe the entire AERE process including all activities that are necessary for the elaboration of realizable requirements concerning all relevant issues (see Fig. 2). In each requirements activity, the elicitation instruction must therefore inform about the capabilities and assumptions of the SPL. By using this information, a requirements engineer is then able to guide the elicitation better, and to initiate negotiation as soon as the expectations of a customer seem to contravene the given SPL constraints.

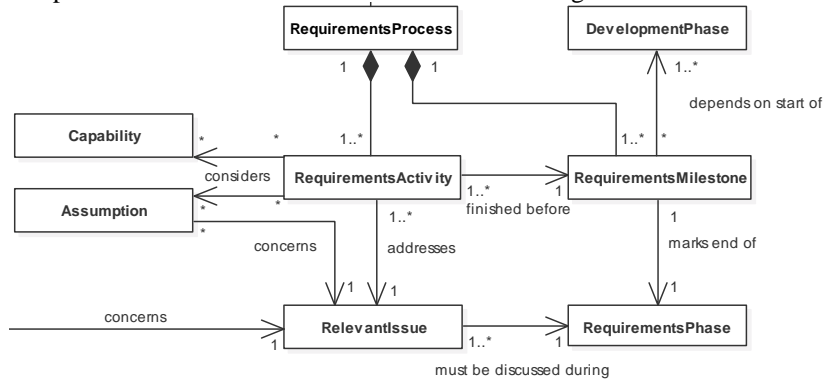


Fig. 2. Structure and relationship of an AE requirements process

In the aforementioned example, the need to know something about workplaces would lead to a corresponding requirements activity in which a requirements engineer has to elicit information about the workplaces at which the system should be used. During this step, assumptions that the SPL makes regarding the nature of workplaces (e.g., visualization power of the devices) is used to check early on whether an appropriate user interface could be developed easily. If not, the requirements engineer is immediately able to start negotiations, or to inform the customer about extra costs, even if concrete requirements have not been anticipated.

However, an open research issue is the right degree of detail the tailoring approach must have. This is actually a problem as some tailoring steps seem to be easily describable in an almost algorithmic manner, while others are hard to prescribe at all.

Furthermore, not only the tailoring approach, but also the resulting elicitation instructions need an appropriate degree of preciseness. In order to reflect the assumptions and relevant issues in a form that is suitable for AE requirements engineers, we have therefore recently performed a survey with RE experts to elicit requirements these elicitation instructions should fulfill (addresses RO 2). Now, we are faced with the challenge on how to implement these requirements in an elicitation instruction template including the definition of section structures, text modules, or content-generating algorithms. This is also a prerequisite to develop a supporting tool.

Regarding evaluation, we have already performed a feasibility study very early in our research in order to show that AERE tailoring is basically possible, and that the effort for performing this tailoring is justifiable (see [9]). Furthermore, we have

calculated possible benefits when using our elicitation instruction in comparison to traditional AERE artifacts through a simulation on how different approaches deal with the mentioned types of elicited requirements. According to this simulation, we expect a possible saving of up to 38% rework effort when using instructions according to our approach. Of course, an open question is whether these improvements are actually achievable in reality. Thus, we still have to confirm these calculations through empirical studies. From a scientific point of view, we plan to evaluate in a controlled experiment that elicitation instructions based on our tailoring approach provide requirements engineers better (more complete and / or faster) with information about a SPL than traditional AERE artifacts do. From a rather practical point of view, we plan to conduct case studies in industry for demonstrating that the use of such (better informing) instructions can also lead to a faster achievement of a satisfactory fit between requirements and SPL characteristics.

4 Conclusion

In this paper, we have presented our current and planned work regarding the problem that a satisfactorily fit between customer requirements and SPL characteristics cannot be achieved sufficiently fast during AE. To cope with this problem, our research aims at providing an AE requirements engineer with better SPL knowledge by means of tailored elicitation instructions. While basic work such as the performance of a feasibility study or the definition of a conceptual model is (almost) done, the development of the actual tailoring approach, the definition of an elicitation instruction template, as well as the performance of a final evaluation is still open.

References

1. Clements, P., Northrop, L.: *Software Product Lines*. Addison Wesley, 2001
2. Deelstra, S., Sinnema, M., Bosch, J.: Product derivation in software product families: a case study. In: *The Journal of Systems and Software*, vol. 74. Elsevier, 2005
3. Rabiser, R., Grünbacher, P., Dhungana, D.: Supporting Product Derivation by Adapting and Augmenting Variability Models. In: *SPLC*. IEEE, 2007
4. O'Leary, P., Rabiser, R., Richardson, I., Thiel, S.: Important Issues and Key Activities in Product Derivation: Experiences from Two Independent Research Projects. In: *SPLC 2009*
5. Djebbi, O., Salinesi, C.: RED-PL, a Method for Deriving Product Requirements from a PL Requirements Model. In: *CAiSE*. Springer, 2007
6. Guelfi, N., Perrouin, G.: A Flexible Requirements Analysis Approach for Software Product Lines. In: *Proceedings of RefSQ 2007*, LNCS 4542. Springer, 2007
7. Baum, L., Becker, M., Geyer, L., Molter, G.: Mapping Requirements to Reusable Components using Design Spaces. In: *Requirements Engineering Conference*. IEEE, 2000
8. Adam, S.: Improving SPL-based Information System Development Through Tailored Requirements Processes. In: *Doctoral Symposium @ RE 2010*
9. Adam, S., Doerr, J., Ehresmann, M., Wenzel, P.: Incorporating SPL Knowledge into a Requirements Process for Information Systems. In: *PLREQ @ REFSQ 2010*. Essen, 2010
10. Wiegers, K.: *Software Requirements – Second Edition*. Microsoft Press, 2005
11. IEEE's Software Engineering Body of Knowledge. IEEE, 2004

Pragmatic Variability Management in Requirement Specifications with IBM Rational DOORS

Ekaterina Boutkova

Group Research and Advanced Engineering

Daimler AG

Wilhelm-Runge-Straße 11

89081 Ulm, Germany

ekaterina.boutkova@daimler.com

Abstract. **[Context and motivation]** Reuse of requirements leads to reduction in time spent for specification of new products. Variability management of requirement documents is an essential prerequisite in terms of a successful reuse of requirements. It supports the decisions whether existing requirements can be reused or not. One possibility to document the variability is feature modelling. **[Question/problem]** A problem for the application of the feature modelling approach in requirements specification is a lack of tool support. An analysis of the existing tools shows some deficiencies that render these tools unsuitable for the needs of Daimler passenger car development (Daimler PCD). **[Principal ideas/results]** An extension of requirements management tool IBM Rational DOORS functions can help to solve the challenge of tool support in many typical cases at Daimler PCD. **[Contribution]** This paper presents a landscape of the requirement specifications at Daimler PCD and a survey of old and new approaches of variability management. Furthermore, the requirements on tool support are discussed and a pragmatic extension of the requirement management tool DOORS are presented.

Keywords: DOORS, feature modelling, requirement, specification, variability, variability management

1 Introduction

Specifications play an important role in the development process of high tech products. They are all-embracing and the specification process itself is very costly in terms of time and effort. The reuse of requirements could help to reduce the time spent for development with regard to variability diversity [3]. Variability management (VM) of requirements is an essential prerequisite in terms of a successful reuse of requirements. A basic approach for requirements reuse in practice, at least at Daimler passenger car development (Daimler PCD), is to tag each requirement for each car model the requirement is valid for. Given the fact that specifications for a single component usually consist of several hundred to several ten-thousand requirements [5] it is obvious that such an approach is labor-intensive and error-prone. One promising possibility to document the variability in specifications is feature modelling [6]. It bases on the assumption that there are product properties (features) that are the drivers for similarities or dissimilarities.

There are not too many tools on the market which support the feature modelling approach and are compatible with DOORS [4]. At the same time DOORS is the standard tool for requirements management in the industry (e.g. automotive domain). DOORS itself has no functions for requirements management for product lines. The analysis of the tools on the market shows that none of these tools fulfils all the defined requirements from Daimler PDC. Thus the decision was made to build DOORS extensions that provide the necessary functionalities to implement a feature driven VM.

This paper presents a lean approach for VM in requirement specifications. This approach builds on feature modelling [6] and is tailored for use with the requirement management tool DOORS both for variability documentation and variability management. The new approach allows creating of a new specification for a product variant within 15-30 minutes. Therefore the novel approach radically reduces the time effort for creating variant specifications.

The next section briefly describes the requirement specifications landscape at Daimler PCD. Section 3 describes approaches for variability management in requirement specifications that have been used so far in most cases in Daimler PCD specifications. Along with the description of the specification landscape in Section 2, this provides a good description of the situation in which an enhanced but lean variability management approach has to work in. Additionally, we explicitly identify some challenges and boundary constraints an enhanced variability management approach has to cope with. Section 4 discusses why available tools were not adequate in our situation and presents our solution for lean variability management within DOORS. An example illustrates the current solution along with its impacts on the requirements engineering work practice. Section 5 presents first experience of the using the DOORS extensions and the feature based variability management (FBVM). Section 6 provides a short conclusion.

2 Requirement Specifications

At present, three abstraction levels of specifications exist at Daimler PCD. These levels are *vehicle specifications*, *system specifications* and *component specifications*.

A *vehicle specification* documents the marketing view (e.g. target customer group, product positioning on the market) as well as the design and technical concepts for the new car as a whole.

A system (as the term is used at Daimler PCD) consists of tangible customer functions and characteristics of these. Typical examples of systems are Outside Light Control or Seat Heating. A system is typically implemented as interplay of many components. Outside Light Control, for example, is implemented by up to 22 ECUs [8]. Thus, a *system specification* builds a bridge between vehicle requirements and component requirements as it decomposes the vehicle-level requirements on the systems and allocates detailed requirements to the contributing components.

A *component specification* is a central technical specification and is basis for tendering and contract with a supplier.

Around 100 system specifications and about 400 component specifications [5] will be created for one new car model. The classification of Regnell et al. [11] places many of these specifications into the category “large-scale RE” (order of 1,000 re-

quirements). Some specifications contain up to 50,000 individual elements (DOORS objects) and therefore fall in the category “very large-scale RE” [7]. Most system and component specifications are written by using DOORS.

3 Variability Management in Requirement Specifications

This section presents the overview of the existing approaches for the VM at Daimler PCD and describes challenges as well as requirements for VM approaches from Daimler PCD. At last, an enhanced approach, which based on feature modelling, will be presented.

3.1 State of the Practice

All approaches presented in this section as well as the whole vehicle development at Daimler PCD are structured through the car models. If a new car model is planned, all existing requirement specifications will be analysed if the reuse of requirements is possible for the needs of the new car series. The analysis of all existing approaches for the variability management in specifications is presented in [1]. This paper gives only a short overview.

ID	REQUIREMENT
1	The light must be activated through the turn down of sun visor [E-Class, S-Class]
2	The décor material of the sun visor must be drapery.
3	The décor material of the sun visor must be leather [E-Class , S-Class]

Fig.1. Attribute Columns Approach.

The documentation of the variants in *the requirement text* is a first possibility. In this case, the variants are a part of a requirement. Figure 1 shows an example. This notation means: The requirement (1) is valid for E- and S-Class but not for A-Class. The requirement (2) is valid for A-, E- and S-Class. The requirement (3) is valid for E- and S-Class but not for A-Class

ID	REQUIREMENT	A	E	S
1	The light must be activated through the turn down of sun visor.	-	YES	YES
2	The décor material of the sun visor must be drapery.	YES	YES	YES
3	The décor material of the sun visor must be leather.	-	YES	YES

Fig.2. Attribute Columns Approach.

The *attribute columns* approach uses attribute columns in DOORS in order to document the variants. Each variant receives one column including the following

possible values, “yes” (the requirement is valid for this variant) or “-” (the requirement is not valid for this variant). Figure 2 shows the example as above. Most of the time, the columns represented new car models. Sometimes, the columns represent the components (e.g. front and rear bumper). Additionally, there were cases where the columns represent a combination of a car model and a component or car body type.

ID	REQUIREMENT	CAR MODEL
1	The light must be activated through the turn down of sun visor.	E-Class S-Class
2	The décor material of the sun visor must be drapery.	common
3	The décor material of the sun visor must be leather.	E-Class S-Class

Fig.3. Multi-Enumeration Value Attribute.

The approach *multi-enumeration value attributes* is a modification of the attribute columns approach. Car series values will be documented in one column with multi-enumeration value attribute. Additionally, there exists the “common”-value, which marks requirements that are valid for all variants. Figure 3 shows the same example as Figure 2.

The *chapter approach* is based on the differentiation between common and the variant-specific requirements. The common (or variants-spanning) requirements are documented in an appointed part of the specification. This part is placed in each specification. If the variant requirements differ from overlapping requirements, they will be documented in the variant specific part.

The analysis of current approaches shows some shortcomings. The main advantage of these approaches are (1) bad scalability, (2) high effort for each new specification document (as all requirements has to be classified in the context of the new specification document), and (3) no support in modelling dependencies between requirements.

3.2 Challenges and Requirements

Hubaux et al. [14] analysed the challenges on feature modelling faced by practitioners. All of the identified challenges on VM [4] match the challenges, which were identified at Daimler PCD [2]. This section exemplifies most important challenges and boundary conditions (i) to (iv).

(i) *Size and Complexity of Variability Models and Specifications.* There are many reasons for variability in one car. One simple component has 10-100 possible variation points. So, one car has some thousands of variation points which must be documented and managed. Another facet is the size and heterogeneity of specifications. Most requirement specifications at Daimler PCD are very large. However, small requirement specifications (e.g. for a standard part as a bolt) also exist. So, the range in specification size is very large. The range between the number of variable requirements as well as the range between the numbers of component or system variants is also very large.

(ii) *Human Factor*: The component and system specifications at Daimler are written by engineers. They are trained experts in the areas of the vehicle, system and component development. They have corresponding education as mechanical or electrical engineers. That is why most of them do not have in-depth know-how in information technology in general and requirement management in specific. Furthermore the time for training is limited. The main task of the engineers is the component or system development. In a typical vehicle project, they spend two to six months on writing their specification document. The next three to five years, the engineers “only” accompanies the development of his component, a job which consists of many tasks – most of them are not related to requirements management.

(iii) *Requirements Management Tool*: The requirements documents at Daimler PCD are documented using DOORS.

(iv) *Specification Language*: A further challenge is that specifications at Daimler PCD are mainly documented in natural language.

During the analysis of the existing approaches for VM in specifications and many interviews with experts at Daimler PCD, the following requirements for a variability management approach were identified:

(R1) There has to be no redundant documentation of the same requirement in the specification or requirement library.

(R2) The requirements that are valid for all variants must be recognizable.

(R3) The mapping between requirement and variant must be understandable for everyone (i.e. especially for non-variability-management-experts).

(R4) It must be possible to compare the different variants in the specification directly.

(R5) The mapping between requirements and product variants must be semi-automatically testable. For example, if there are two requirements that are mutual exclusively, there must be an effective way to check this.

(R6) The variability management approach must be scalable, i.e. it must be applicable to specifications that consist of several ten-thousands of requirements.

The next section describes the Feature Based Variability Management (FBVM) approach.

3.3 Feature Based Variability Management

The result of the analysis of current approaches at Daimler PCD shows that an improved approach for variability management in specifications is needed. The literature research shows that the orthogonal [9] feature modelling [6] has a high potential for variability documentation of the natural language specifications. Feature modelling is already applied at Daimler PCD in some projects for variability management during design and implementation phases. The adoption of feature modelling for variability in specifications could help to have a continuous variability management at all development phases.

The basic principle of feature modelling is to show not only the variable characteristics of a product (optional or alternative features) but also the common characteristics of all products from the same product line (mandatory features).

The *feature modelling* was introduced by Kang et al. [6] as a part of Feature Oriented Domain Analysis (FODA). The main idea of the feature modelling is to create an abstraction level above requirements in order to support visualization, documentation, and the communication between stakeholders of product variants. FODA defines features as “*user-visible aspects or characteristics of the domain*”[6]. In this paper, feature is defined as *a characteristic of one product of the product line, which it could have*, i.e. a feature does not necessarily have to be user-visible. Architectural properties, for instance, could also be a feature. This modelling technique was refined in many researches and is highly accepted in industry [8]. There are many variations of feature modelling approaches, which were developed to solve different problems in various domains [8]. The exact description of the feature modelling technique is not the aim of this paper and could be taken from [6, 8].

During the analysis of the existing approaches for the VM in specifications, three levels of need were identified at Daimler PCD.

- At the first level, only the variable features (optional or alternative features) must be documented. There are no relationships between features. The number of features is low (about two to seven parent features, each with two to five leaf features).
- At the second level, there are some relationships between features. The number of features is middle (about two to 15 parent features, each with two to ten leaf features).
- At the third level, all features (common and variable) must be documented. There are many relationships between features. The number of features is high (about 30 to 50 parent features, each with ten to 30 leaf features).

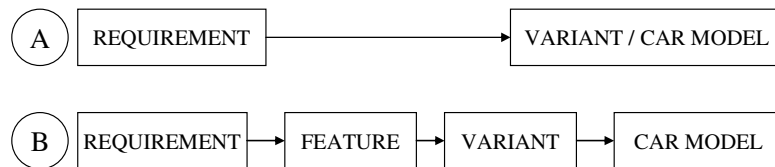


Fig.4. Mapping of Requirements.

An important precondition for the application of FBVM in specifications is the necessity of a paradigm change. The idea is the separation of requirements and car models (Fig. 4A). In case of feature modelling, the requirements should be mapped to features (Fig. 4B). This mapping from requirement to feature creates a non-recurring effort, because in most cases it does not change anymore. A new variant (e.g. a new car model) is specified by including or excluding of features from the variant model.

FBVM approach was developed considered to the needs of all development departments (e.g. mechanics, electrics and electronics). The different values of the methodical needs (e.g. the electronic components have much more features and relationships between features as mechanic components) were defined and the corresponding stages of the methodical support were work out. FVBM approach fulfils all requirements R1 to R6 on the VM approach from Section 3.2.

4 Feature Based Variability Management with DOORS

This section describes at first the shortcomings of the tool support in the area of variability management. Then the requirements on tool properties will be presented. Section 4.2 presents the extensions of DOORS functions, which enable to use DOORS for variability management in specifications. At last, the application of the enhanced variability management approach – which is supported through DOORS extensions – is presented.

4.1 Feature Modelling Tools

There are not too many tools on the market which support the feature modelling approach. DOORS is the base requirement tool at Daimler, so it is understandable that a first requirement for VM tool is compatibility with DOORS. There are three tools on the market which fulfil this requirement: Gears from BigLever [12], pure::variants [14] and metadoc [13]. The further requirements for tools are:

(TR1) The mapping between requirement and variant must be easily understandable. This means that no code or Boolean operator can be used for this mapping.

(TR2) The structure of a specification may not be changed by a VM tool.

(TR3) The comparison of different variants in the same requirement specification must be possible inside this specification.

(TR4) The variability model may not be changed from the specification.

At last, a wish of most users was to have the possibility to document and manage variability in the same tool as the requirements are documented in, i.e. inside DOORS.

None of the above mentioned tools fulfils all requirements (TR1-TR4). Especially for the first two levels of complexity (see Section 3.3), intense trainings and the introduction of a new tool or new specification structures seemed to be not adequate. Thus we decided to build some DOORS extensions that provided the necessary functionalities to implement a feature driven variability management.

For more complex situations (which could be characterized as complexity level three), we decided to invest in trainings and to use a tool that is capable to deal with complex feature models. Here, we selected pure::variants.

4.2 Variant Module in DOORS

The *variant module* is an extension of DOORS functionality. It provides the possibility to document and manage the variability within DOORS. These extensions were programmed in the DXL and Java programming language [15]. The base of the variant module is one formal module and two link modules in DOORS. The formal module documents the variability. The first link module contains links inside of formal module. The impact of the link is dependent on the object type (AND, OR, NOT, IF-THEN). The second link module supports the mapping between the variant module and specification modules.

Table 1. Object Type Values

Attribute	Description
Source-ID	This attribute gives the unique number for each object in the variant module
Object Type	<p>This attribute makes it possible to document the features and the relationships between these features. The possible values of this attribute are:</p> <ul style="list-style-type: none"> - A <i>criterion</i> denotes the parent features. - A <i>criteria value</i> denotes the leaf features. Each feature has at least two leaf feature (exist and not exist). - A <i>combination</i> is the selection of criteria values considering the rules. The combination can be created through links, which connect (AND) the required criteria values. - A <i>configuration</i> is the selection of criteria values and/or combinations considering the rules. The configuration can be created through links, which connect (AND) the require criteria values and /or combinations. - A <i>negation</i> is an auxiliary construct for the logical operator NOT. The out-link from a negation object shows on the negated criteria value. - A <i>group</i> is an auxiliary construct for the logical operator OR. - A <i>rule</i> describes the relationship between two criteria values. The in-link from rule denotes the condition (IF) and the out-link from rule denotes the effect (THEN).
Object Text	This attribute makes it possible to document the names of features, combinations, configurations and rules.
Object Short Text	This attribute makes it possible to document the user abbreviation for names of features, combinations, configurations and rules.
Visible	This attribute has binary states {false, true} and makes it possible to manage which features and which configurations will be available in the specification.
Simple Formula	This attribute shows the formal composition of combination, configuration and rules.
Variant Document	Variant Document denotes the first object of variant module. This is the precondition for the mapping with specification modules.

The variant module has six columns (which are the representation of the corresponding attributes) for the variability documentation. These attributes are presented in Table 1. The variant module template consists of four parts, which are: features, combinations, configurations, and rules. In the “feature part” all features can be documented. In the “combination part” valid combinations of features can be documented. In the “configuration part” valid combination of features, which are the required variants of components or systems, can be documented. The difference between combination and configuration is that a combination could be a part of a configuration. Therefore, a configuration can consist of features and combinations.

Additionally, the variant module supports the mapping from requirements to features and provides the following functions:

- Propagate the feature mapping from parent requirement to a child requirement.
- Propagate the feature mapping from child requirement to parent requirement.
- Propagate the feature mapping from parent requirement to all children requirements.

4.3 FBVM with Variant Module in DOORS

The application of the FBVM approach in DOORS will be described with the sun visor specification as example. The sun visor was chosen for this case study because its specification is not too large. There are about 400 technical requirements. In the initial specification all requirements are mapped to the next three car models.

The first step is to identify both the common and the variable requirements. Afterwards, it is possible to identify the features, which are the rationales for the requirements variability. Currently this process bases on manual reviews.

The second step is the feature documentation in the variant module. The parent features (e.g. completion, hinge bearing) have the *Object Type Value* "Criteria". The leaf features (e.g. simple, double) have the *Object Type Value* "Criteria Value". The hierarchic relationships between parent features and leaf features are documented with the standard DOORS object hierarchy. The "needs" relationships between features are documented with rules. Fig. 5 shows the variant module for the component sun visor.

ID	Object Type	Voltage 5.5 - Variantenmodul	Visible	Short Text	Simple Formula
SV_V-1	Variant Document	1 Variantenmodul			
SV_V-18		1.1 Features			
SV_V-2	Criteria	1.1.1 Completion			
SV_V-4	Criteria Value	Completion Simple		simple sv	simple sv
SV_V-5	Criteria Value	Completion Double		double sv	double sv
SV_V-3	Criteria	1.1.2 Hinge Bearing			
SV_V-7	Criteria Value	Hinge Bearing Simple		simple hb	simple hb
SV_V-8	Criteria Value	Hinge Bearing Double		double hb	double hb
SV_V-19	Criteria	1.1.3 Counter Bearing			
SV_V-20	Criteria Value	Counter Bearing Simple		simple cb	simple cb
SV_V-21	Criteria Value	Counter Bearing Double		double cb	double cb
SV_V-23	Criteria	1.1.4 Light			
SV_V-24	Criteria Value	with Light			with Light
SV_V-25	Criteria Value	without Light			without Light
SV_V-10	Combination Cluster	1.2 Combination	False		
SV_V-12	Combination	Simple Sun Visor	False		(simple sv) AND (simple hb) AND (simple cb)
SV_V-16		1.3 Rules	False		
SV_V-17	Rule	Rule 1	False		IF (simple sv) THEN (simple hb)
SV_V-26	Rule	Rule 2			IF (simple sv) THEN (simple cb)
SV_V-27	Rule	Rule 3			IF (double sv) THEN (double hb)
SV_V-28	Rule	Rule 4			IF (double sv) THEN (double cb)

Fig.5. Sun Visor Variant Module in DOORS

The next step is the mapping of requirements to features. At first, the specification module must be mapped to the variant module. Then, features from the variant module must be imported into the specification. The features will be imported as attribute with multi-enumeration values (equal to leaf features names). For each feature, a column will be created. The requirements that are valid for all features do not have

any mapping to features. The variable requirements must be mapped to correspondent features. Fig. 6 shows the mapping in the sun visor specification.

ID		Completion	Hinge Bearing	Counter Bearing	Light
1	1 Sun Visor				
14	This is a common requirement.				
2	This is a requirement on simple sun visor.	simple sv			
3	This is a requirement on double visor.	double sv			
4	This is a common requirement.				
5	This is a requirement on sun visor with light.				with L
11	This is a common requirement.				
6	This is a requirement on sun visor without light.				without
7	This is a requirement on simple counter bearing.			<input checked="" type="checkbox"/> simple cb	
8	This is a requirement on double counter bearing.			<input type="checkbox"/> double cb	
12	This is a common requirement.				

Fig.6. Specification Module

If a new variant of the sun visor is required, a new configuration must be created in the variant module. Fig. 7 presents the new variant of the sun visor. It has simple completion, simple hinge bearing, simple counter bearing and no light. It is now possible to create the variant specific sun visor specification in only a few minutes. In the specification module, the filter function must be activated. The result of this filtering can be saved as variant view which represents the wanted specification.

SV_V-14	Configuration Cluster	1.4 Configuration	
SV_V-15	Configuration	A-Class (Serial)	(Simple Sun Visor) AND (without Light)

Fig.7. New Configuration in Variant Module in DOORS

5 Discussions

This section presents the first the experiences with the application of the FBVM in specifications at Daimler PCD as well as the experience with the variant module in DOORS.

An important step during the introduction of the FBVM was the definition of all relevant terms (e.g. *feature*, *variant*, and *product*). It was necessary because a common language is the precondition for the successful collective work of many people with different education and practice experiences.

The development of the migration process from *attribute based approaches* to the FBVM approach was a challenge. This transition is currently done manually. Especially costly in terms of time is the manual identification of the features. This process takes about one day for an average component specification document. Therefore, a semi-automatic feature identification approach was developed and is currently being evaluated.

In order to manage the complexity of vehicle variability, a solution with *decentralized variability models* was chosen. In the *decentralized variability models* approach a variability model for each component and each system must be created. The decentralized variability models allow to reuse the requirements and configurations for individ-

ual components and systems in a short time. The support of the configuration process at the vehicle level as well as the synchronisation of variability models at all levels are the next challenges.

Particular attention was spent on the acceptance of the new VM approaches from engineers and support group. In order to win the acceptance, multiple presentations of the FBVM approach were offered. Furthermore, the pilot applications were supported not only through the support group but also through the developer of the FBVM experts. This course of action allowed collecting experience with this new approach and to find the weaknesses and possible improvements.

The first experiences with the FBVM are positive. FBVM is well accepted by the pilot users. The pilot users at the component level develop mechanical components with few features (not more than ten features) as well as electronic components with about 70 features and 10^{16} theoretical possible variants. At the system level, the pilot users develop systems with about ten features. The main advantages of FBVM are:

(1) *The reduction of the time effort for creating the variant specifications.* The creation of a specification for the new car model with the *attribute column approach* for a specification with 1.000 requirements takes about 8 hour (it means 30 second for analysing and mapping per requirement). This mapping must be repeated for each new car model. The introduction of the FBVM for a specification takes about 8 hour for the identification of features and the building of the variability model. This effort is nonrecurring in contrast to the mapping of requirements to car models. The creation of a specification for the new car model with FBVM for a specification with 1.000 requirements takes about 15 minutes.

(2) *The know-how about reasons for the variability.* The features show the rational for the existence of variability. During the application of the *attribute column approach* this information was hidden.

(3) *Independence from a specific requirements management tool.*

One disadvantage of FBVM approach is the effort for the training of the engineers and support groups. The training consists of the presentation and the building of the variant module and it takes about five hours. The further disadvantages are the initial effort for the creation of the variability models and the necessary adjustments of the processes following the specification phase.

The first experiences with DOORS extensions are positive, too. These extensions allow documenting and managing the variability within DOORS. Therefore, an engineer doesn't need to use different tools. Most of the variant modules are not too large and consist of not more than ten parent features. Many analysed component and system variability models at Daimler PCD match this condition. In other cases, the same variability management approach can be implemented using the tool pure::variants [14].

6 Conclusion

This paper presents pragmatic extensions of the requirement management tool DOORS. These extensions support a new FBVM approach for the documentation and management of variability in natural language specifications. The extensions of DOORS functions make it possible to document and manage the variability in requirements with DOORS.

The presented DOORS extensions are suitable for smaller numbers of features with no complex relationships between them. The typical well supported number of features is about ten parent feature and five leaf features. Many analysed component and system variability models at Daimler PCD match this condition.

The FBVM approach and extensions of DOORS functions do not solve all challenges on VM in specifications. But this approach including the presented DOORS extensions make it possible to effectively and efficiently reuse requirements in specifications at Daimler PCD right now. New specifications for new product variants can be configured from the corresponding feature models within a few minutes.

There are some related works, which try to manage the variability in specifications or in technical documentations. Nicolás et al [10] gives a good overview about existing approaches for VM in specifications. The main difference to all another approaches is that the presented approach allows keeping the existing specifications language, the existing structure of specifications as well as the requirements management tool.

References

1. Boutkova, E.: Variantenmanagement in Anforderungsdokumenten: State of the Practice (Variability Management in Requirement Specifications: State of the Practice). In: Softwaretechnik -Trends 9(1) (2009) in German
2. Boutkova, E.: Herausforderungen für Variabilitätsmanagement in Anforderungsdokumenten (The Challenges for Variability Management in Requirement Documents). PIK 2010, Paderborn (2010) in German
3. Cheng, B.H.C., Atlee, J.M.: Research Directions in Requirements Engineering. In: Future of Software Engineering (FOSE'2007), IEEE (2007)
4. Hubaux, A., Classen, A., Mendonca, M., Heymanns, P.: A preliminary review on the application of feature diagrams in practice. In: Proceeding of VaMoS 2010, 53-59 (2010)
5. Houdek, F.: Challenges in Automotive Requirements Engineering. Industrial Presentations by REFSQ 2010, Essen (2010)
6. Kang, K.C., Cohen, G., Hess, S.J.A., Novak, W.E., Spencer, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Peterson (1990)
7. Leuser, J., Ott, D.: Tracking Semi-automatic traceability in large specifications. In: R. Wieringa and A.Pesson (Hrsg.): REFSQ 2010, LNCS 6182, 203-217 (2010)
8. Lichter, H.; von der Maßen, T.; Nyßen, A.; Weiler, T.: Vergleich von Ansätzen zu Feature Modellierung bei der Softwareproduktlinienentwicklung (The Comparison of Feature Modelling Approaches in the Software Product Line Development), (2003), in German.
9. Pohl, K.: Requirements Engineering, dpunkt Verlag, Heidelberg, 2007.
10. Nicolás, J., Toval, A.: On the generation of requirements specifications from software engineering models: A systematic literature review. Information and Software Technology, 51(9), Elsevier, 1291–1307 (2009)
11. Regnell, B., Svensson, R.B., Wnuk, K.: Can we Beat the Complexity of very Large-Scale Requirements Engineering? In: Paech, B., Rolland, C. (eds.) Requirements Engineering: Foundation for Software Quality. LNCS, vol. 5025, 123 (2008)
12. BigLever Gears <http://www.biglever.com/>
13. Metadoc <http://www.metadoc.de/anforderungsmanagement/werkzeuge/fm/73-feature-modeller>
14. Pure::Variants http://www.pure-systems.com/pure_variants.49.0.html
15. SpryLab <http://www.sprylab.com/>

Prioritizing Requirements: An Experiment to Test the Perceived Reliability, Usability and Time Consumption of Bubblesort and the Analytical Hierarchy Process

Geurt Johan van Tuijl, Wouter Leenen, Zhengru Shen,
Inge van de Weerd, Sjaak Brinkkemper

Department of Information and Computing Sciences, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{g.j.vantuijl, w.a.leenen, z.shen}@students.uu.nl
{i.vandeweerd, s.brinkkemper}@cs.uu.nl

Abstract. Software vendors often face the difficult task to deal with large amounts of requirements that enter the company every day. When dealing with this vast amount of requirements, the notion of deciding which requirements will be addressed first, and which will be addressed later, is an important decision. To support software development teams in decision-making, different prioritization techniques are discussed in previous literature. In this paper, two existing prioritization techniques called Analytical Hierarchy Processing (AHP) and Bubblesort are investigated in order to measure their outcome in terms of usability, time consumption and perceived reliability. By conducting an experiment among Dutch Master students, we discovered that Bubblesort outpaced AHP on all aspects, although this could not be supported statistically. However, based on our findings, we can conclude that there is a strong indication that Bubblesort is considered favorable compared to AHP even though it receives less attention in current literature.

Keywords: Requirements prioritization, Bubblesort, Analytic Hierarchy Processing, Software Product Management

1 Introduction

Software vendors have to handle the requirements that enter the company through customers, sales & marketing, and research & development every day. Typically, customer demand exceeds the pace at which software is developed [1], usually causing the final release to be outdated before it has been published. Eliciting the required information from customers can be difficult to achieve, especially when multiple customers with diverse expectations are involved [2]. Customer involvement is therefore a major contributing factor to company success [3].

An important issue when dealing with this vast amount of requirements is the notion of deciding which requirements are being addressed first and which will be addressed later. This difficulty, i.e. the prioritization of requirements, is recognized as

an import activity in product development [4, 5] and product management [6, 7]. The process for prioritizing software requirements faces the difficult task to be simple and fast, however, also needs to provide accurate and trustworthy results [8]. A solution to this problem can be found in existing techniques such as the Binary Priority List [1], Analytical Hierarchy Process [9] and Hierarchical Cumulative Voting [10]. Regardless of which technique is carried out by the user, each technique should be able to support the product manager in decision-making but also needs to decide which requirements are the least important.

1.1 An Overview of Requirements Prioritization Techniques

Researchers, as well as professionals in industry, have proposed many requirements prioritization methods and techniques. Different techniques have been discussed considerably and compared with others in experimental studies [11], empirical studies [4, 12] and literature studies [17]. According to a review proposed by Racheva et al. [13], these techniques could be classified into two main categories: techniques that are applied to small amounts of requirements (small-scale) and techniques that scale up very well (medium-scale or large-scale). Examples of small-scale techniques include round-the-group prioritization, multi-voting system, pair-wise analysis, weighted criteria analysis, and the Quality Function Deployment approach. Among medium-scale or large-scale techniques, the MoSCoW technique, the Binary Priority List, the Planning Game and the Wieggers's matrix approach are frequently used. Considering the sometimes complex algorithms used in existing techniques, often supporting tools are needed to deal with large amounts of requirements.

Another classification for requirements prioritization techniques is given by Berander et al. [14]. Similar to Racheva et al. [13], they also divided existing techniques into two main categories: (1) techniques which assume that values can be assigned, by an expert, to different aspects of requirements and (2) methods that include negotiation approaches in which requirements priorities result from an agreement among subjective evaluation by different stakeholders. Examples of techniques that apply to the first category are Analytical Hierarchy Process (AHP) [9], Cumulative Voting, Numerical Assignment, Planning Game and Wiegger's method. An example of the second category would be the Win-Win approach.

The prioritization technique that is popular among companies and researchers is the AHP technique. AHP is a multiple criteria decision-making method that has been adapted for prioritization of software requirements [15, 7]. In a software project that has n requirements, a decision maker requires $n(n-1)/2$ pair-wise comparisons. Advantages of AHP are that it is trustworthy and has a strong fault tolerance. Along with AHP, another technique called Bubblesort [16] is one of the simplest and most basic methods that have been used for requirements prioritization and is similar to AHP in many ways [17]. However, Bubblesort receives little attention in literature compared to AHP even though both perform well according to other studies [15]. A problem that arises for both AHP and Bubblesort is the scalability. As soon as the number of requirements exceeds twenty, the prioritizing procedure takes too long. However, solutions such as structuring the requirements in a hierarchy of interrelated

requirements [15], or applying machine learning techniques [17] have been proposed to overcome this problem.

The variety of prioritization techniques makes it difficult to select the most appropriate one for a specific project situation. Still, the prioritization of requirements proves to be a fundamental part of the responsibilities for a software product manager but it can be a quite demanding, difficult and time consuming task [17, 18]. Even small software product companies that often deal with smaller amounts of product requirements, could benefit from structured techniques to guide them to the best possible software product releases.

1.2 Problem Statement

Concluding from a research of Karlsson et al. [15], Bubblesort and AHP performed the best compared to four other prioritization techniques in terms of reliability, ease of use and fault tolerance when dealing with relatively small amounts of requirements. The reason to center research upon AHP and Bubblesort is because both perform really well according to other studies [15]. Still, there is significantly more literature devoted to AHP. Furthermore, both techniques can easily be applied on a small amount of requirements, which makes it easier to use in an experiment setting [21]. As stated, AHP receives a lot of attention whereas Bubblesort does not, and for this reason, we will discuss both techniques and elaborate on the specific differences between them by providing an experiment with the aim to extend current literature on Bubblesort and AHP. The application of the techniques will be performed by assessing and comparing the results of both techniques through a prioritization experiment with students at a large university in the Netherlands. This experiment will provide specific information about the perceived reliability, time consumption, and usability for both techniques. The research question we want to answer is:

How do the requirement prioritization techniques Bubblesort and Analytical Hierarchy Process perform when compared to each other in terms of perceived reliability, time consumption, and usability?

The next five sections are structured as follows. Section 2 presents the applied research approach and provides a clear explanation for this choice. Section 3 provides information about the execution of the experiment. Section 4 provides a brief explanation of how to interpret the results, and follows with the actual results for the variables tested for each of the techniques. Section 5 includes the overall conclusions. Finally, section 6 provides a discussion, goes into the limitations of this research, and some suggestions for future research.

2 Experimental Design

The goal of this experiment is to compare two prioritization techniques, AHP and Bubblesort, in terms of usability, time consumption and perceived reliability.

2.1 Subjects

For this experiment, we provide twelve students from Utrecht University, The Netherlands, with a list of requirements for a new version of Google Maps. Each subject uses Google Maps regularly. Furthermore, each subject has knowledge of the importance of requirements prioritization and has approximately the same age in order to maintain integrity and diminish variable influences on the results. The subjects did not receive any credits or reward for their participation.

2.2 Objects and Instrumentation

We created a list of twenty requirements¹ for a widely used web-based software package, called Google Maps. Google Maps is a web mapping service application and technology provided by Google. It allows the user to navigate anywhere on earth to view satellite imagery, maps, terrain, 3D buildings and street views. The reason for selecting Google Maps is because it is a well-known web based software product and thus widely-used on a daily basis, causing users to have different experiences and deficiencies about the application.

Each subject is provided with the list of requirements and with Excel based custom-made tools for AHP as well as Bubblesort. Both tools provide the subject with a graphical representation of the algorithm used behind AHP and Bubblesort. Since both tools were made in an Excel environment, any possible influences caused by the spreadsheets can counterbalance each other. Along with the tools, subjects were provided with instructions on how to use both tools.

2.3 Data Collection and Analysis Procedure

Data is collected by means of a questionnaire. The time consumption is registered through identifying the exact starting time and the exact ending time of the students excluding any breaks in between. The data is collected in an informal experiment setting at the university, where the subjects use the provided excel based tools along with the questionnaire. When the subjects are finished, the data is entered manually in SPSS (a statistical analysis tool) in order to measure predefined outcomes.

To perform an effective and meaningful comparison of both requirements prioritization techniques, we take three evaluation factors into account: (1) *time consumption*, which is defined as the interval between the time the user starts prioritizing the requirements and the time when the user is finished prioritizing [21] using either of both prioritization techniques; (2) *ease of use*, which is measured by means of direct questions to each participating student concerning complexity and number of comparisons; and (3) *perceived reliability*, which is defined as the degree to which the final requirements ranking differs from the ideal target ranking. In this research, the ideal target ranking is the ranking the subject has in mind based upon

¹ The list of requirements can be found at <http://people.cs.uu.nl/weerd/reew-appendixa.pdf>

implicit knowledge. This ideal ranking per subject is the ranking the student made manually based upon implicit knowledge and without the use of a tool or technique.

2.4 Validity Evaluation

In this experiment, students are used as subjects instead of businessmen; hence reasoning and interpretation of requirements might not be representative for software product companies [20]. Still, the students do have a thorough knowledge about the software product and the presented requirements at hand, and therefore, reasoning and interpretation threats should influence the results only minimally.

On an individual level, there is the risk for students being influenced by familiarity with the requirements or learning from their experience with the first prioritization method upon switching techniques. However, by counterbalancing the groups we tried to minimize this threat. Furthermore, the subjects of the experiment could be influenced by fatigue. Hence, we limited this threat by keeping the requirements understandable and the number of requirements low.

3 Execution

Before starting with the experiment, the subjects were randomly divided into two groups. Each subject that is part of group A will apply Bubblesort on the list of requirements whereas the subjects from group B will apply AHP. When both groups are finished, they will switch techniques, i.e. subjects from group A will then apply AHP and subjects from group B will then apply Bubblesort. The reason for alternating the techniques is to take away any habituation influences of the subjects on the results; i.e. any advantage gained by using one technique before the other for the whole group. In addition, if all subjects start with the same technique, they might be able to influence the results of the other technique due to time constraints or time consumption. When both groups are finished, each subject will fill in a brief questionnaire. This questionnaire encompasses the evaluation of both tools by assigning a grade to the perceived reliability, the usability, and the time consumption of both tools. Afterwards, the results of each subject will be analysed.

4 Analysis of the Results

The results from the questionnaire can be found in Table 1. Each subject that took part in this experiment is assigned to a group. Subjects 1 to 6 were part of group A whereas students 7 to 12 were placed in group B. The requirements prioritization process started simultaneously, where group A started with the Bubblesort tool (the abbreviation BS in Table 1) and group B started with the AHP tool. When both groups were finished, each student had to fill in a brief questionnaire with respect to their experiences towards both tools.

In Table 1, each number in the table represents a grade (on a scale of 1-10) that the subject assigned towards the tool. For instance, subject 1 assigned an 8 to the reliability for AHP and a 4 for the reliability of Bubblesort. In addition to these grades, the consistency ratio is also taken into account for the AHP tool. Since this ratio only applies to the AHP tool, this ratio is not available for the Bubblesort tool. The consistency ratio is presented in the AHP column between the two brackets and describes the consistency of the prioritization process, i.e. how consistent was the subject when prioritizing requirements. According to Saaty [9], a ratio of less than 0.10 is considered acceptable. Furthermore, the time consumption (in minutes) is measured throughout the prioritization process to indicate how much time it took to complete the prioritization process. The results are located within the AHP and Bubblesort columns underneath the overarching 'time consumption' column.

Table 1. Questionnaire results, specified per student and group

	Reliability		Ease of use		Time consumption		Overall score	
	AHP	BS	AHP	BS	AHP	BS	AHP	BS
Group A								
Subject 1	8(.172)	4	5	3	6(31)	2(59)	8	6
Subject 2	8(.093)	7	7	6	5(66)	6(42)	8	6
Subject 3	7(.054)	9	8	9	8(14)	7(16)	7	8
Subject 4	5(.023)	7	5	9	5(33)	5(29)	5	8
Subject 5	7(.039)	8	6	4	7(29)	6(35)	7	7
Subject 6	7(.040)	8	6	9	5(75)	9(40)	7	9
<i>Average</i>	<i>7(.070)</i>	<i>7,2</i>	<i>6,2</i>	<i>6,7</i>	<i>6(41)</i>	<i>5,8(37)</i>	<i>7</i>	<i>7,3</i>
Group B								
Subject 7	5(.075)	7	2	4	10(30)	9(20)	4	6
Subject 8	6(.019)	9	7	9	4(60)	9(30)	6	8
Subject 9	6(.144)	8	7	9	5(57)	8(30)	6	8
Subject 10	8(.152)	10	6	9	6(24)	9(18)	6	9
Subject 11	7(.061)	7	7	8	3(25)	6(13)	6	7
Subject 12	8(.061)	4	5	6	4(30)	5(30)	6	4
<i>Average</i>	<i>6,7(.085)</i>	<i>7,5</i>	<i>5,7</i>	<i>7,5</i>	<i>5,3(38)</i>	<i>7,7(24)</i>	<i>5,7</i>	<i>7</i>
Average	6,83	7,33	5,92	7,08	5,67	6,75	6,33	7,17

In addition to Table 1, a graphical overview is presented in the form of a bar graph in Figure 1. In the following subsections, we will elaborate upon each measurement.

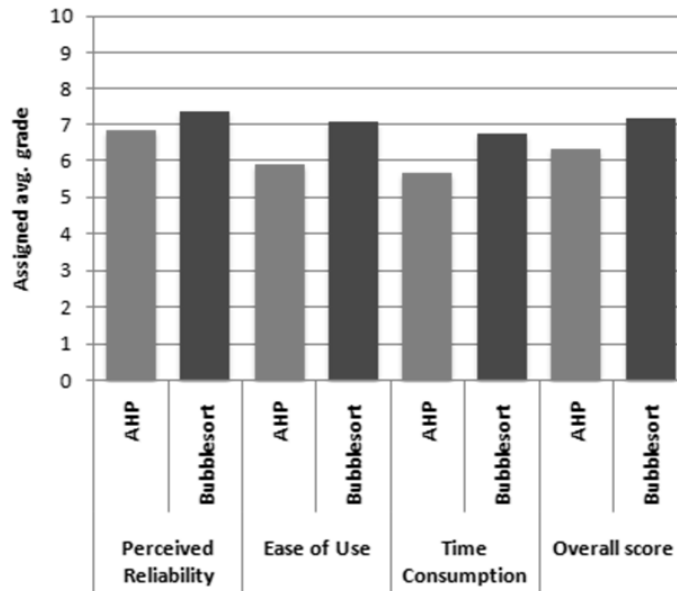


Figure 1. Graphical overview of the results, divided per measurement

4.1 Perceived Reliability

The most left column of Table 1 shows the obtained reliability scores from the subjects involved in the experiment. These scores, on a scale of 1 (low) - 10 (high), were assigned by each subject based on their consensus with the actual ranking obtained through each technique. Furthermore, these results should be considered as 'perceived' because the results are based on the basic reasoning process of non-actual stakeholders in the development of this Google Maps release. When testing for significance between both groups A and B, using the Mann-Whitney Test, there proved to be no significant difference between both groups ($\alpha = .05$). Therefore, group A did not score significantly better than group B and thus we can merge both group scores and assess the full group result.

Before continuing with the evaluation of the reliability scores, the consistency of the AHP technique needs to be taken into consideration. As can be seen from table 1, the contents of the brackets, next to the assigned reliability score for AHP, provide the Consistency Ratio (CR) for the valuations provided by the subjects. This consistency ratio is composed of the consistency index and the random index [7, 19]. Any CR lower than .10 is considered acceptable, however, according to Saaty [9] these 'acceptable' scores are hard to obtain. Looking at the results, consistency proves to be very good with only three people obtaining CR-scores higher than .10. For one particular subject we observed a remarkable consistency rate in AHP for filling in the matrix cells, indicating a very inconsistent valuation (.172) of the requirements, while

his appraisal of the obtained ranking (8) was quite high. For that particular subject we can conclude that by providing a very inconsistent valuation of the requirements, his results should be considered unreliable.

Using the Wilcoxon Signed Ranks test, the overall deviation between the scores assigned by the subjects for AHP (avg. 6.83) and Bubblesort (avg. 7.33) proved not to be significant ($\alpha = .05$). Nevertheless, the difference between the mean of the scores of the perceived reliability for the two techniques suggests that Bubblesort provides better results on 'perceived' reliability.

4.2 Ease of Use

Ease of use (i.e. the usability) is measured through the use of a questionnaire. The subjects were kindly asked to rate the usability of Bubblesort and AHP on a scale from 1-10, where 10 is the highest score.

When calculating the average score of the usability of both tools, we see that Bubblesort has a mean of 7.08, whereas AHP does not reach beyond 5.91. This result is as we expected since the Bubblesort tool required less time to complete, and is therefore regarded as easier to use. This conclusion could also be derived from the fact that group A started with Bubblesort first. Since group A did not have a clue how the AHP tool looked like, they gave Bubblesort an average of 6.7. Group B, which started with AHP first, gave Bubblesort an average grade of 7.5. Since the tool of AHP perceived to be more complex in use, the subjects subsequently found the Bubblesort tool easier to use and thus assigned a higher grade to it. The usability score of both techniques could be influenced by the created Excel-based tools. During the prioritization process, some subjects faced difficulties with respect to the completion of the Excel document whereas other subjects did not face any troubles. Nonetheless, considering both tools were made with Excel, any possible drawbacks caused by the limitations of spreadsheet software during the prioritization process would likely counterbalance each other, thus negating their effect

Analyzing the usability between group A and group B, based on the Mann-Whitney Test ($\alpha = .05$) we again did not find a significant difference between both groups. Since no significant difference was found, we decided to merge both groups A and B. When we subsequently use the Wilcoxon Signed Ranks test to test for significant differences between the scores of AHP and the scores of Bubblesort, we again did not find any significant differences ($\alpha = .05$). Although the average scores deviated, based on our sample size, this deviation cannot be assigned to the population yet. There is only a strong presumption that Bubblesort might be easier to use than AHP.

4.3 Time Consumption

The averages in Table 2 show that Bubblesort performs better compared to AHP in terms of the actual time consumption. The results from start/end time were recorded by the subjects. As illustrated in Table 2, the difference in time consumption (sample mean) of the two prioritization techniques is 9.3 minutes, which is relatively large.

Even though both medians are nearly the same (30.5 for AHP and 30.0 for Bubblesort); the difference in time consumption can easily be noticed when taking standard deviation into account. In our experiment, the distribution of the actual time consumption is not skewed and there is no extreme outlier as well. Hence, we can conclude that the Bubblesort tool is more time saving compared to the AHP tool.

Aside from measuring time consumption in minutes, we use the time consumption score, which is a subjective judgment given by the subjects after they use both techniques. The higher the score, the less time consuming the tool is. As shown in Table 2, Bubblesort achieves a higher score, which means that the subjects believe Bubblesort is less time consuming. The mean score that Bubblesort obtains from the subjects is 6.7, which is higher than the 5.7 of AHP, and the median of the scores for these two techniques are 6.5 and 5, respectively. Although, according to the standard deviation, the scores obtained by Bubblesort are more dispersed than AHP, this does not change the fact that there still is an indication Bubblesort is a better technique with respect to time consumption.

Table 2. Time consumption in minutes and grade per tool

Method	Median	Mean	Std. deviation(SD)
AHP (in minutes)	30.5	39.5	19.5
Bubblesort (in minutes)	30	30.2	12.9
AHP (in grade)	5	5.7	1.9
Bubblesort (in grade)	6.5	6.7	2.2

4.4 Overall Rating

After taking reliability, ease of use, and time consumption into account, we also asked the subjects to assign an overall score to both techniques. This was again based on a scale of 1-10; the higher the score, the higher the appreciation of the technique.

If we look at the overall scores in Table 1, we notice a higher score for Bubblesort in comparison to AHP. This is coherent with the other scores obtained for reliability, ease of use, and time consumption, because those also indicated better averages for Bubblesort. The average score for AHP is 6.33, while Bubblesort shows an average score of 7.17. Furthermore, subjects were asked to explain their overall score and indicated that the combination of the quickness and the reliability of Bubblesort, in comparison to AHP, were the main reason for assigning higher scores to the former.

Analyzing the overall scores for both techniques within group A and group B, based on the Mann-Whitney test, did not show any significant differences ($\alpha = .05$) when comparing for Bubblesort but there was a minimal significant deviation between group A and B for AHP. This is considered to be quite remarkable since the overall score should be derived from the reliability, ease of use, and time consumption. Still, merging of both groups was difficult and a general comparison would be less reliable. Nevertheless, overall comparison, although not encouraged here, showed no significant deviation between Bubblesort and AHP.

5 Conclusion

Throughout this research, we attempted to provide an answer to the following research question:

How do the requirement prioritization techniques Bubblesort and Analytical Hierarchy Process perform when compared to each other in terms of perceived reliability, time consumption, and usability?

For this research we conducted an experiment among twelve Master students of a large University in the Netherlands to test the applicability of Bubblesort and AHP. The subjects used both techniques in prioritizing twenty fictional requirements for a new software product release of Google Maps. The perceived reliability, ease of use and time consumption was assessed by comparing both techniques. Taking both groups into account, there were some distinct differences found.

On average, Bubblesort scored better than AHP in all aspects of the comparison. The perceived reliability, the time consumption and the usability of Bubblesort was greater than AHP. Nevertheless, the division of both groups did have one minor implication on the results: the overall valuations of the tools, assigned by the subjects, seemed to be influenced by the group division.

During the experiment we tried to eliminate any advantages gained from using one of the tools before the other. When looking at the results for both groups, we saw no further significant differences between both groups ($\alpha = .05$). Therefore, we can conclude that both groups scored the same on both techniques.

After merging both groups to create an overall average, we can conclude that Bubblesort still scores better than AHP for perceived reliability of the results. Furthermore, Bubblesort also scored better on time consumption and ease of use by a decent margin. Unfortunately, the sample size proved to be an obstacle in obtaining significant results.

Concluding from our research, Bubblesort scores overall better compared to AHP. There is a strong indication that when the sample group size will be increased, Bubblesort would probably still outpace AHP when both techniques are applied in a similar situation.

6 Discussion, Limitations, and Further Research

Throughout our experiment we measured the perceived reliability, ease of use, and time consumption for both requirement prioritization techniques AHP and Bubblesort. However, the research presented in this paper has certain limitations. These limitations predominantly originate from the size of our research sample and the unavailability of established tools for both techniques.

Firstly, due to a lack of availability of subjects as well as time constraints, we were not able to increase the size of the sample. The time provided to assemble a representative large group of subjects proved to be limited. Therefore, the reliability of the obtained results can be influenced. Upon further review, the small sample size was not considered to be a normal distribution and non-parametric tests had to be

performed. However, the results seem to give a decent indication of expected results when the sample size is increased. Therefore, we would suggest further research with an increased sample size.

Secondly, the unavailability of a usable and implementable tool for both techniques could have influenced the results. Currently, there is no established tool for Bubblesort available. Therefore we developed a custom Excel-based tool for Bubblesort fitting the number of requirements used within our experiment. Furthermore, we tried to use the IBM© focal point™ tool for AHP but encountered some problems with implementing it. Therefore, we developed a custom Excel-based tool for AHP too. Although this eliminates any problems on comparing the two techniques, there is a possibility that the tools were not comprehensive and sufficient enough due to a lack of validation of the tools. Since the participating subjects did not experience any difficulties nor did we hear any complaints, we believe the tools were sufficient enough to obtain reliable results on perceived reliability. However the results of this experiment on time consumption and ease of use will be very tool dependent and therefore specific for our custom tools.

A final note should be made on the issue of prioritizing large volumes of requirements. In this experiment, a set of twenty requirements was used. For software vendors dealing with small amounts of requirements, Bubblesort may be a good solution to prioritize requirement and reach consensus among the stakeholders. However, in many real-life settings, the amount of requirements easily exceeds the amount of requirements we used. Further research should be done to tool support in which Bubblesort is integrated with proposed solutions as structuring the requirements in a hierarchy.

References

1. Bebensee, T., Weerd, I. vd., Brinkkemper, S.: Binary Priority List for Prioritizing Software Requirements. In: Requirements Engineering: Foundation for Software Quality. LNCS, vol. 6182/2010, pp. 67--78. Springer Heidelberg (2010)
2. Wiegers, K.: First Things First: Prioritizing Requirements. *Software Development* 7(9), (1999)
3. Kabbedijk, J., Brinkkemper, S., Jansen, S.: Customer Involvement in Requirements Management: Lessons from Mass Market Software Development. In: 17th IEEE International Requirements Engineering Conference, pp. 281--286. Atlanta (2009)
4. Lehtola, L., Kauppinen, M., Kujala, S.: Requirements Prioritization Challenges in Practice. In: Proceedings of 5th International Conference on Product Focused Software Process Improvement, pp. 497-508. Japan (2004)
5. Lehtola, L., Kauppinen, M.: Empirical evaluation of two requirements prioritization methods in Product Development Projects. In: Proc. European Software Process Improvement Conference, pp. 161-170. Trondheim, Norway (2004).
6. Bekkers, W., Weerd, I. van de, Spruit, M., Brinkkemper, S.: A framework for process improvement in software product management. In: A.Riel et al. (Eds.): EuroSPI 2010, CCIS 99, pp. 1--12 (2010)
7. Weerd, I. van de, Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, L.: Towards a reference framework for software product management. Proceedings of the 14th International Requirements Engineering Conference, Minneapolis/St. Paul, Minnesota, USA, 319-322 (2006)

8. Karlsson, J., Ryan, K.: A cost-value approach for prioritizing requirements. *IEEE Software* 14(5) 67--74 (1997)
9. Saaty, T. L.: *The Analytical Hierarchy Process*, McGraw-Hill (1980)
10. Berander, P., Jönsson, P.: Hierarchical Cumulative Voting (HCV) - prioritization of requirements in Hierarchies. *International Journal of Software Engineering and Knowledge Engineering* 16(6), 819--849 (2006)
11. Karlsson, L., Thelin, T., Regnell, B., Berander, P., Wohlin, C.: Pair-wise comparisons versus planning game partitioning – experiments on requirements prioritisation techniques. *Empirical Software Engineering* 12(1), 3--33 (2007)
12. Karlsson, J.: Software requirements prioritizing. In: *Proceedings of 2nd International Conference on Requirements Engineering*, pp. 110–116 (1996)
13. Perini, A., Ricca, F., Susi, A.: Tool-supported Requirements Prioritization: Comparing the AHP and CBRank Methods. *Information and Software Technology* 51(6), 1021--1032 (2009)
14. Racheva, Z., Daneva, M., Buglione, L.: Supporting the Dynamic Reprioritization of Requirements in Agile Development of Software Products. In: *Proceedings of the Second International Workshop on Software Product Management 2008, Barcelona*, pp. 49--58 (2008)
15. Berander, P., Andrews, A.: Requirements prioritization. In: A. Aurum, C. Wohlin (Eds.), *Engineering and Managing Software Requirements*, Springer, (2005)
16. Karlsson, J., Wohlin, C., Regnell, B.: An evaluation of methods for prioritizing software requirements. *Information and Software Technology* 39(14--15), 939--947 (1998)
17. Aho, A. V., Hopcroft, J. E., Ullman, J. D.: *Data structures and Algorithms*. Addison-Wesley, Massachusetts (1983)
18. Avesani, P., Bazzanella, C., Perini A., Susi, A.: Facing scalability issues in requirements prioritization with machine learning techniques, In: *Proceedings of 13th IEEE International Conference on Requirements Engineering*, IEEE Computer Society, Paris, France, pp. 297–306 (2005)
19. Firesmith, D.: Prioritizing Requirements. In: *Journal of Object Technology* 3(8), pp. 35--47 (2004)
20. Random RI for AHP, <http://www.docin.com/p-23136057.html>
21. Berander, P.: Using Students as Subjects in Requirements Prioritization. In: *International Symposium on Empirical Software Engineering (ISESE'04)*, pp. 167--176 (2004)

Speed Creation Session: A way to increase the productivity of experts in projects and assure quality requirements

Matthias M. D. Pohle¹, Sven Krause² and Andreas Rusnjak³

¹ Swisscom (Switzerland) Ltd, Corporate Business P.O. Box, CH-3050 Berne, Switzerland

² Zühlke Management Consultants Ltd, Wiesenstrasse 10a, CH-8952 Schlieren, Switzerland

³ Christian-Albrechts-Universität zu Kiel, Department of Informatics, 24118 Kiel, Germany

matthias.pohle@swisscom.com, sven.krause@zuehlke.com, aru@informatik.uni-kiel.de

Abstract. In large and distributed companies the knowledge is spread. Experts come from different locations and subsidiaries. People often work in several projects simultaneously. This leads to an efficiency loss and long product development cycles. What if there is a way to get rid of dozens bilateral and specialized division workshops and preparation meetings? We call our approach “Speed-Creation”. The starting point is fuzzy frontend of innovation, where the rough product or software idea is given. But still it’s vague and fuzzy. At this point the experts ask “What are your requirements and what is the impact on my field of work?” But these answers don’t exist at this point. So we set up a small team being led by a speed-coach. During 72 hours they only work on this single project without distraction. After this seed phase a feasibility study can be easily set to prepare a development project and ultimately production of an innovative product. This paper describes the Speed Creation approach and reports on experiences of applying the method.

Keywords: Project Management, Product Management, Product Development, Innovation Management, Early Phases, Business Analysis, Business Engineering, Agile Development, Team Building, Team Excellence.

1 Introduction

In today’s time we live in a world of information overflow. A lot of development projects have huge scopes and big teams. Often project leader work together with 40 - 50 or even more project delegates and they do not know each other and have never worked together before. The early phase of these kinds of projects is often critical and inefficient. It takes some time to get all the resources and then these 40 to 50 highly specialized delegates rattle the project leader with dozens of specific questions and asking about detailed requirements (see Fig. 1). But the project is in an early phase; the project idea is still fuzzy. Product managers analyzed the idea well. Product managers got the needs, developed an approach and thought about the benefits and if the approach could be substituted by others. So they are on a good path and have made great analytics. But the implementation approach is very general at this time; it is just a rough idea and detailed requirements, e.g. for an ensuing feasibility check of the project and state compliance are still missing.

Due to this the Speed Creation will be introduced as a lightweight process model to gather a lot of substantial requirements by an interdisciplinary group in a short time.

But where is the difference to other creativity groups, e.g. the Focus Groups? Yes, other creative methods proceed similarly and contain many similar components, but why invent something new, if there is an existing working process or similar approach? Nevertheless there are differences. The first difference is focus. The speed creation focuses itself on the requirements that are needed for the implementation. That means the question “What is needed to realize the product?” will be answered. Speed Creation builds that bridge from a rough idea to an 80% draft of business requirements to allow a project team to start with a fundamental base of requirements into a feasibility study. It is a new agile project method that combines the best elements from other methods, for example scrum, kaizen and customer experience design.

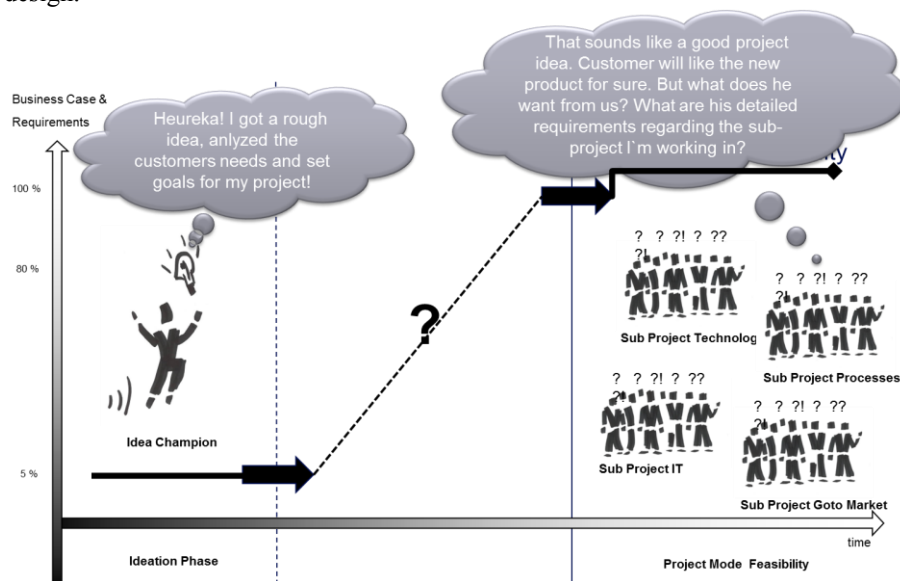


Fig. 1. The problem of exploration projects is showing the gap in the early project phase.

2 Speed Creation process

Speed Creation helps to increase the efficiency of requirements elicitation and consolidation. The method is introduced by describing necessary preparations, how Speed Creation is performed, and how its results then can be utilized (c.f. Tab. 1).

The starting point is an idea. This idea can be developed within an idea workshop [1], a process for generating innovation candidates [2] or any other approach. If you have a good idea, move on to the Speed Creation mode. To ensure that only good ideas make it to Speed Creation, we typically apply a two-step filter. The idea is summed up on one page. During this process, you will describe in a high abstraction the needs of the customer, your approach, the benefit for the customer and your company and the situation of your competitors. This method is called NABC (Need, Approach, Benefit and Competition) [3, 4].

The product manager is responsible for filling out the NABC. This NABC is assessed by a selected management team. Criteria for the approval are the compliance to the product strategy, to the product portfolio and project management. After a positive decision, the Speed Creation can start.

Speed Creation process overview	
Goal	Increase of the productivity of the project members Speed up projects
Precondition	An approved idea for a product or a service development
Steps	1) Workshop preparation and set up a small and interdisciplinary project team
	2) Deepening and concretization of the project order (prepare ecosystem, goals, scope)
	3) Create a first view of the offering
	4) Specify use cases along the “customer experience chain”
	5) Define general requirements, requirements for others (like other products or projects)
Post condition	<ul style="list-style-type: none"> • 80% of the business requirements, which is the basis of the ensuing feasibility study • A video where the product manager (or the project team) presented the customer needs and product goals • A motivated core team with a common understanding • A cultivated cross-organizational teambuilding

Tab. 1. Speed Creation - Overview

There is a fixed set for workshops, analyzing and documentation methods we use. Every Speed Creation can be somewhat different, hence needs to be planned individually. That is one of the key success factors. In Speed Creation we set up a small and interdisciplinary project team (see Fig. 2). It should not be bigger than 5-7 people. We have a rough idea and start to fill white pages of paper with requirements. And in a creation workshop the critical mass is round about 7 people. You can easily do a review with 10+ people, but you can hardly create new coherent content with big groups. So we set up this interdisciplinary team and focus on an intense 72 hours' workshop. The interdisciplinary team consists of product manager, marketing manager, sales people, business analyst and IT or technical people. Within this workshop we analyze and concretize the idea in depth. But it is still in best guess mode. We create the input which the experts will refine later.

When starting the Speed Creation we look at the ecosystem (on the market as well as internally, e.g. it-systems) first. Then we set goals for the outcome of the speed creation. Due to this it is also very important to specify the scope of the idea which will be developed in a Speed Creation. The objective of this phase is to bring the needs with the goals and the scope in agreement.

A matrix ensures that each goal covers at least one need. Additionally, a second matrix ensures that each scope element covers at least one goal. That gives us the certainty that we cover all needs in further considerations. With this we ensure that we do not observe goals or scope elements which have no impact on the customers' needs. Thereby we ensure that the stakeholders do not exploit the situation for their own goals.

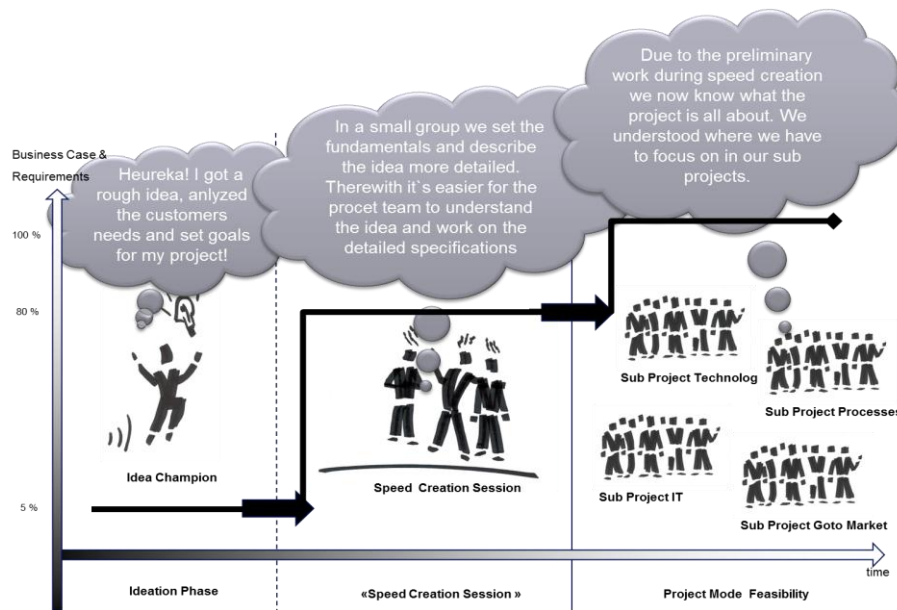


Fig. 2. The “new” story showing how speed-creation closes the gap in the early project phase.

In a workshop we create a draft for the offering to the customer. Later we specify a first draft of use cases along the so called customer experience chain. In this process we use the typical use case notation based on UML and the table-specification after Cockburn [5]. The customer experience chain separates the customer interactions with the enterprise into specified phases. The 8 phases are notice, inform, order, take in operation, use, pay, get support and offer change.

The documentation within a Speed Creation is much hands-on and must be done very quickly (on the fly) during the teamwork, e.g. writing requirements into a word document, taking snapshots of drawings on whiteboards/ flipcharts. But hands-on does not mean a bad quality of requirements. It only means that missing details have to be later supplemented or out-formulated. The requirements described are based on a phrase pattern [6] in order to ensure understandability. Basis of the documentation is a structure similarly to Volere [7] and IEEE 1233-1998 [8]. The structure contains the chapters' situation analysis (customer needs, goals, and scope), business case (cost effective study), commercial offer, use cases, general requirements, and requirements for others (like other products or projects).

The outcome of a Speed Creation contains the content of what the requirements are. Maybe some details are missing, but all the 40 to 50 project delegates who are involved in the project, later in the feasibility phase, get a good overview on what is planned on a more detailed basis. Within the ensuing feasibility study the delegates now know what to work on; which problems they need to solve and where to dig deeper into their expertise. They then add the detailed requirement specifications.

The speed philosophy claims to improve projects by focusing: Take one piece of work at a time. Concentrate on one topic and then move on to the next. Do not do too much of parallel work streams. Second is to work out of office where none of your daily colleagues can distract you from the project work. Also it is nice to work in a pleasant environment maybe at a lakeside. Third you got a moderator who is a professional on how to do product development. The expertise regarding the specific product will not come from method knowledge, but the moderator brings in the method knowledge.

3 Specifics to the Speed Creation process

A good time management especially during Speed Creation workshops is not easy but important. You have to define time blocks for each topic area and communicate the existing time to execute the topic area. It is important that the persons involved can focus themselves on the essence of the topic. Based on a clear structure given by the Speed Creation document or prepared flipcharts which helps you to save time, the moderator needs a good feeling for the situation. The moderator has to decide whether he would like to adapt the given time. But, the moderator has to ensure that the duration for the other topics is sufficient.

During the workshop we use flipcharts and drawings. The pictures get pasted into the document. It is only a first draft to refine on later. There is no need to draw it exactly in a software tool. There is time in a speed-creation. Nevertheless every picture is described in prose by the speed-documenter.

4 Lessons Learned

There are four big wins if you choose to use a speed-creation in your project. You will speed up your projects, you create a common understanding and conserve it in the documentation, you ease feasibility and you cultivate cross organizational teambuilding.

Projects will be faster and more efficient because you got a detailed first guess of requirements within 72 hours and not 5-6 months. The Modus operandi comprises a good time management for the different workshops, including preparation and documentation. For a solid time management, professional preparation is important.

The common understanding gets more important the bigger the project team gets in the later phases, e.g. the feasibility or implementation phase. The concentrated work we create at the same location facilitates a common understanding. One can achieve a common understanding by supporting discussions and exchange opinions. Naturally it happens formally during workshops, but also informally during e.g. dinner. The importance of an informal exchange outside the scheduled workshops is not to be underestimated. So for that reason, Speed Creation workshops should be held outside the office including overnight stays in hotels.

The ensuing feasibility project will be easier. All the delegates who come into the project later on to provide expertise on specific topics now know exactly what to do. The frame is set; they can pick up on the existing requirements and fragments and detail them any further.

Due to the fact that the speed-creation core-team is multidisciplinary we tear down walls between different organizations and lead the people into a teambuilding phase just for the product. This works well because of the 72 hours approach. Within that time people work hard and play hard. So they start to be a team.

5 Key Success Factors

There were six key success factors which determine to have an efficient Speed Creation. First there was expectation management. How concrete or fuzzy was the idea itself? Was it clear what the idea is? Were the goals set? If not, you could hardly have a good Speed Creation. Second was to customize Speed Creations. Each project differs, so the detailed workshop plan needs to be individualized. 80% are mostly identical, but some parts just need to be individualized. Factor three and four were the concentrated work without distraction and to get feedback from a jury. During a Speed Creation people were so focused that they sometimes tend to be over focused. Therefore we presented what they have done during the day to a jury. This jury consists of colleagues who did not take part on the workshop process. They get the presentation and give feedback. One feedback should be positive (what they liked in the presentation), another should be negative, but please stay constructional. Factor five was having a multidisciplinary Speed Creation team. And finally factor six the product manager and/or business engineer. During a Speed Creation the product manager and/or business engineer might focus on his role and concentrate on the discussion. Everything else is up to the two speed coaches who moderate and also document. But after a Speed Creation the product manager has to take the lead and move on with his business case and the business requirements set. The outcome from the Speed Creation is still an 80% draft and it needs to get finalized.

References

1. Maiden, N., Gizikis, A., Robertson, S.: Provoking Creativity: Imagine What Your Requirements Could Be Like. *IEEE Software* 21, 5 (2004) 68-75
2. Gorschek, T., Fricker, S., Palm, K., Kunsman, S.: A Lightweight Innovation Process for Software-Intensive Product Development. *IEEE Software* 27, 1 (2010) 37-45
3. Carlson, C., Wilmot, W.: *Innovation: Five Disciplines for Creating What Customers Want*. Crown Business (2006)
4. A shortened English version of Kirstine Vinderskov's work paper 'NABC – metoden, www.learnship.eu/media/2657/dk%20nabc.doc
5. Cockburn, A.: *Writing Effective Use Cases*. Addison-Wesley Longman (2000)
6. Cohn, M.: *User Stories Applied: For Agile Software Development*. Addison-Wesley Longman (2004)
7. Robertson, S., Robertson, J. (1999). *Mastering the Requirements Process*. Addison-Wesley
8. IEEE (1998). *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Standard 830-1998

Author Index

A

Adam, Sebastian	19
Ahmed, Sharmin	13

B

Boutkova, Ekaterina	25
Brinkkemper, Sjaak	37

C

Chowdhury, Muhammad Iftekher	13
------------------------------	----

F

Ferrari, Remo	13
---------------	----

K

Kof, Leonid	1
Krause, Sven	49

L

Leenen, Wouter	37
----------------	----

M

Madhavji, Nazim	13
-----------------	----

P

Penzenstadler, Birgit	1
Pohle, Matthias M. D.	49

R

Rusnjak, Andreas	49
------------------	----

S

Shen, Zhengru	37
---------------	----

V

Van De Weerd, Inge	37
Van Tuijl, Geurt Johan	37