(Semi-) Automatic Merging of Product Variant Requirements Specification Documents

Martin Beckmann[1] Anna von Pestalozza[2]

Abstract: Due to the increasing complexity of systems, the difficulty to create high-quality requirements specification documents has increased as well. A possibility to deal with this challenge is the reuse of requirements. An industry partner has multiple separate specification documents for its different product variants. To make use of the existing information one of the documents is copied and used as a basis for the new product variant. As a result of the reproduction of only a single specification document, the exclusive content of other documents is not included. In this paper, we propose an approach to automatically create a single specification document out of the documents of the product variants, in order to exploit the full potential of the existing data. This is achieved by a comparison of the document entries and a categorization based on the comparison. The resulting document is supposed to be used as an overview over system features as well as the basis to automatically extract content for new product variants. Also we define criteria to identify inconsistencies between the documents. The approach is evaluated by merging two specification documents and manually analyzing the result. Out of 2,552 regarded entries, there are 75 entries that were handled incorrectly.

Keywords: Specification Document Merge, Requirements Reuse, Textual Requirements Elicitation

1 Introduction

The reuse of requirements has been a research topic for quite some time [LMV97]. Although it has been recognized repeatedly as an efficient method to improve the development of systems and software [GB15, EBB09], there are still many challenges in making existing requirements reusable [Ch12]. To address these challenges a vast number of different approaches have been proposed [LJB98], especially in relation to software product lines [Al10]. Nevertheless, in industry this knowledge is still insufficiently implemented [Ka15]. As a consequence, the most widespread techniques in industry are still based on copying (i.e. cloning) existing data and manually modifying it [PFQ14, RCC13].

This non-systematic approach leads to multiple separate (but very similar) requirements specification documents for each product variant [Kn02]. A major disadvantage becomes apparent when a new product variant is developed: One has to choose between a number of existing similar requirements specifications. As a result, available information contained in other documents is not included. We aim to facilitate the specification process by proposing

[1] Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, martin.beckmann@tu-berlin.de
[2] Helmut-Schmidt-Universität Hamburg, Holstenhofweg 85, 22043 Hamburg, anna.grvpestalozza@hsu-hh.de

an approach that supports the creation of a specification document containing all information of other product variants. In the automotive industry a fictional car containing the features of all product variants is denoted as a *150% car* [Gr14]. Hence, the purpose of our approach is to create a 150% requirements specifications document.

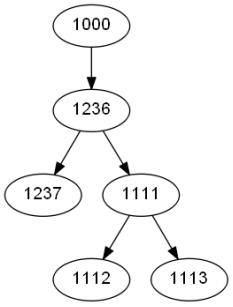
What the approach basically does, is to compare the entries of the requirements specification documents. In case a unique entry is found it needs to be added to the 150% requirements specification. If the entry already exists, it needs to be checked whether it has been modified. Since the documents are primarily manually created, inconsistencies between the documents may occur. We present several criteria to find possible inconsistencies. We use the approach to merge two requirements specification documents of product variants of one system. The merging process left 75 entries of 2,552 entries assigned incorrectly.

2 Background

In this section we present the kind of requirements specification documents our approach uses. In order to ensure atomic requirements and to provide traceability, requirements management tools often present requirements in the form of tables [FE00]. A small excerpt of a specification document is shown in Fig. 1a.

ID	Text	Level	Type
1000	**1. Section**	1	Heading
1236	**1.1 Subsection**	2	Heading
1237	Descriptive Text	3	Information
1111	Requirement Text	3	Requirement
1112	Req Refinement	4	Requirement
1113	Req Refinement	4	Requirement
	....		

(a) Excerpt of a requirements document



(b) Tree structure of the excerpt

Fig. 1: Example of a requirements specification and its corresponding tree structure

The document consists of multiple rows. Each row represents an entry in the document. Each entry has a number of attributes (columns). In Fig. 1a the attributes *ID*, *Text*, *Level* and *Type* are displayed. Besides these attributes the document may contain more attributes. These are necessary for further development tasks. However, they are not necessary for the approach and hence not displayed in the example.

The *ID* makes each entry of the document unambiguously identifiable. This is needed to enable further capabilities of requirements management tools such as traceability [So12, p. 113]. In this work we consider the *ID* as a unique string, which is manually assigned to each entry. Most requirements management tools automatically assign *IDs* to objects. The

manually assigned *IDs* stay the same in case a document is copied. As a result, it can be used to find out whether an entry already exists in another specification document. The *text* attribute contains the actual requirement text. The *Level* is necessary to create a document structure. On the first *Level* in Fig. 1a is a section. This section may consist of multiple subsections which are placed one level below the according section. A section or subsection can contain further entries which are also placed one level below. This may also be used to refine a single requirement as it is shown with the entry with the *ID 1111* which is detailed by the refinements with the *IDs 1112* and *1113*. The resulting structure corresponds to an ordered labeled tree [TC12]. The corresponding tree to the document in Fig. 1a is shown in Fig. 1b. The *Type* attribute describes what kind of entry it is. In the excerpt the entries denoting a section and subsection are of the type *heading*. The entries *1111*, *1112* and *1113* are of the type *requirement*. The entry with ID *1237* shows that specification documents not only contain the actual requirements but also information parts [Th11] which also have to be considered. Therefore, we use the term entry instead of requirement.

3 Related Work

Since the structure of a requirements specification document can be considered as a model, the merging of documents is closely related to the merging of models [Br06]. Because of the underlying model of requirements management tools, it is possible to make additional assumptions. These assumptions concern properties of the tree-like structure as it has been shown in section 2. This structure also distinguishes the situation from merge algorithms that work on unstructured (e.g., line-based) inputs such as source code [ALL12].

The merging of different development artifacts has been recognized as a necessity and hence received a lot of attention by the computer science research community [Me02]. In that sense merging is often perceived as an activity in the context of version control. As a result, it is assumed that there are two versions which originate from one common source. There is published work on this kind of merging for structured requirements specifications [We91]. In contrast to the mentioned work, our situation differs in the sense that we do not have an originating document. The approach uses documents as an input which might have been changed simultaneously and no common source can be identified. To the best of our knowledge, we are not aware of a work that addresses this specific situation.

4 Automatic Requirements Specification Document Merging

Before merging multiple requirements specification documents into a single document, it is necessary to assure the compatibility of the attributes of each document. This compatibility applies to a number of aspects as attributes may have different data types (e.g., integers, strings, enumerations). First of all, it is necessary to ensure that the merged document contains the attributes of all documents which are included. Attributes which are unique

in a document can be added. This step is not necessary, if the requirements management tool supports entries within a document with differing attributes (e.g., ProR, DOORS Next Generation). Besides the unique attributes, there may also exist inconsistent attributes. We consider attributes to be inconsistent, if they have the same name but are not equal. This concerns deviating data types as well as different characteristics in case they have the same data type. The latter includes different limits (integers), length boundaries (strings) and values (enumerations). Some of the cases can be handled automatically (e.g., adjusting limits of integer attributes). Other cases (e.g., deviating attribute data types) cannot be handled automatically without loss of information or the risk of introducing new inconsistencies.

4.1 Merging of Requirements Specification Documents

The actual merging of the requirements specification documents starts by choosing one document. The chosen document is copied. The copied document is the designated 150% requirements specification (henceforth denoted as $RS_{150\%}$). The current document whose content is added to the $RS_{150\%}$ is denoted as RS_{add}. The entries of $RS_{150\%}$ are compared with the entries of the remaining documents based on the *ID* attribute. This comparison yields two possible outcomes: The considered entry already exists in the document or it cannot be found. In the former case, it is necessary to find out whether the entry is different from the existing entry. If it cannot be found, it needs to be added. The comparison is implemented as pairwise comparison. Since the requirements specifications we consider typically consist of a few thousands entries, the pairwise comparison is not a limiting factor.

Unique Entries. If an entry is found in RS_{Add} that does not exist in $RS_{150\%}$, it needs to be added to $RS_{150\%}$. Since the meaning and understandability of a requirement is highly related to its context, it must be placed in an appropriate position in the document. We suggest to insert new entries after their predecessors. The predecessor of an entry is the entry on the same level before the entry itself. An advantage of this approach: if there are multiple unique entries in a row, their order is maintained. This is the case, because the predecessor of each entry is added to $RS_{150\%}$ before and can hence be used for placement.

Fig. 2a and Fig. 2b each show an excerpt of a specification with unique entries. Adding the additional entries of Fig. 2b to the excerpt in Fig. 2a results in the situation in Fig. 2c. The entry with ID *7* was placed after the entry with ID *4* and entry *9* was placed behind its predecessor entry *2*. Entry *8* remains behind its original predecessor entry *7*. In case the excerpt in Fig. 2b is designated as the $RS_{150\%}$ the resulting order differs. In terms of model merging this means that the operation is not associative. We argue that this does not impact the use of the $RS_{150\%}$ as the affected entries are most likely independent of each other.

In case there is no predecessor (the first entry below a parent entry), it is not possible to use a predecessor to place an entry. As the first entry often provides explanatory information, we suggest to place it either again as the first entry below the common parent entry or as the second entry. For the latter case it is assumed that $RS_{150\%}$ already contains an explanatory

ID	Text	Level
1	**1. Section**	1
2	Used Signals	2
3	Signal 1	3
4	Signal 2	3
5	Req 1.1	2
6	Req 1.2	2

(a) Excerpt of a document

ID	Text	Level
1	**1. Section**	1
2	Used Signals	2
3	Signal 1	3
4	Signal 2	3
7	Signal 3	3
8	Signal 4	3
9	Req 1.3	2

(b) Excerpt of a document

ID	Text	Level
1	**1. Section**	1
2	Used Signals	2
3	Signal 1	3
4	Signal 2	3
7	Signal 3	3
8	Signal 4	3
9	Req 1.3	2
5	Req 1.1	2
6	Req 1.2	2

(c) Possible result

Fig. 2: Merging of unique elements into a common document

entry as the first entry. In any case, necessary adjustments of the order require less effort and provide less potential of error than inserting entries manually. The reason for this is that the copying of all (potentially hundreds) attributes of an entry is more error-prone than moving an entry.

Categorization of Existing Entries. Aside from unique entries, one has to consider entries that exist in multiple documents. As requirements specifications evolve, their entries are subject to changes. As a consequence, it is necessary to include entries that may already exist but have changed over time. We need to distinguish three different categories:

- Duplicates. Entries that are identical.
- Variants. The entries differ, but still describe the same issue.
- Unrelated entries. Entries that have no relation to one another.

We separate the entries into the different categories by using a string comparison. The problem of using string comparison on natural language text is that it will definitely result in wrong categorizations, due to the ambiguity of natural language [BKK03]. For instance, the correction of spelling mistakes or adjustments of parameters are minor string changes. Nevertheless they may significantly change a requirement. Hence, the changed entries should definitely be included in the $RS_{150\%}$. Extensive string changes on the other hand may not change the actual content at all.

There is a plethora of work about identifying requirement duplicates by the means of natural language processing (i.a., [FCC13, Da01]). In contrast to the aforementioned work our situation differs, since the entries with the same ID are originally copies of another and thus should be very similar. Because of the high similarity the choice of the similarity measure does not heavily impact the result. Hence, we decided to use the edit distance algorithm *Jaccard* as it provided good results in aforementioned works and is easy to implement. To

separate the three categories we need two textual thresholds. The value θ_D is supposed to separate the duplicates and variants. The value θ_V is supposed to separate the variants and the entries that are unrelated. The resulting conditions are shown in equation 1.

$$100\% \geq \theta_D \geq \theta_V \geq 0\% \tag{1}$$

If the similarity of two entries is above θ_D the entry is identified as a duplicate. If the similarity is below θ_V the entry is considered unrelated. If the similarity is below θ_D and above θ_v it is considered a variant. It may be possible to assign the thresholds θ_D and θ_V the same value, since we use a second criteria for the distinction of entries. As stated before, the position of an entry in the document structure plays an important role for its understandability. In consequence, we assume that the similarity of the position of an entry relates to its meaning and hence indicates whether two entries with the same *ID* are still related. The position of an entry is the path consisting of the parent nodes from the entry itself to the section it belongs to. For instance, for the entry *1113* in Fig. 1b the position is defined by the path *1113*, *1111*, *1236*, *1000*. The positional similarity is determined using these paths as input. As a similarity measure the *Jaccard* algorithm is applied again. In addition to the textual thresholds, we use the positional thresholds η_D and η_V. The aim of the positional similarity is to refine the results of the textual comparison. The corresponding relation of the positional thresholds for the categorization is shown in equation 2.

$$100\% \geq \eta_D \geq \eta_V \geq 0\% \tag{2}$$

As with the textual similarity the thresholds η_D and η_V can be the same. For the categorization to be unambiguous either the thresholds for textual or the positional similarity must be set distinctively. The actual values for the thresholds are determined in the evaluation. How an entry is further processed depends on its category. Duplicates are discarded. Entries considered unrelated need to be examined manually. Those entries deemed as variants need to be placed accordingly – same as the entries that are added. We suggest to append those entries one level below the original entry and mark them as related but differing.

4.2 Inconsistencies

As the specification documents are mainly created and maintained in a manual manner, it may happen that existing *IDs* do no longer belong to entries they belonged to before the cloning of a document. In that case the approach may not find new entries or place them wrongly. To mitigate this issue we define a number of criteria which indicate that an entry may not have the correct *ID*. For the sake of brevity we restrict our presentation of the categories to an explanation and examples. Inconsistent *IDs* might affect the addition of entries as well as the categorization of entries. Every entry that matches our criteria of an inconsistency is not included in the document and must be treated manually.

Unique Entries. For the unique entries, it is not possible to deduce problems by comparison, since there are no entries to compare them to. Instead we use the position of the entries.

ID	Text	Level
1	**1. Functions**	1
2	**1.1. Function x**	2
3	Req 1	3
4	Refinement 1.1	4
7	Refinement 1.2	4
8	Req 2	3
9	Refinement 2.1	4

(a) Excerpt of a document

ID	Text	Level
10	**3. Contact Person**	1
2	**3.1. Function x**	2
11	Person A	3
12	Person B	3

(b) Excerpt of another document

Fig. 3: Entry with ID *2* in different positions of the specification documents

An example of a problematic situation is displayed in Fig. 3. Both documents contain an entry with the *ID 2* and the same text. In Fig. 3a the function is supposed to contain requirements of the function. In Fig. 3b it is supposed to contain the responsible persons of the function. Assuming the entries below the functions are unique, the addition of the entries to the other document would result in a wrong placement (e.g., requirements should not be placed among contact persons). As the sections on the first level of the document represent a very basic outline of the document, we concluded that an entry must stay within the same section as before. Hence, we classify every entry which is supposed to be placed below or after an entry in a different section to belong to the category **Inconsistency I**.

As a consequence, further unique entries may lack a reference for their own placement. For instance, *Refinement 1.1* in Fig. 3a would be placed below its original parent entry. Since *Req 1* was not included, it cannot be placed accordingly. We define these entries to be in the category **Inconsistency II**.

Existing Entries. In case the *ID* of an entry is found in another document, it is possible to use the found entry for comparison. If the textual similarity is below the threshold θ_V, we consider the entries to be unrelated to one another. The rationale is that a major textual change may indicate a change in regard to content as well. As a consequence, with the value x for the result of the textual comparison, **Inconsistency III** is defined as follows:

$$\textbf{Inconsistency III}(x) := \theta_V > x \tag{3}$$

Analogous for the value y representing the result of the positional comparison **Inconsistency IV** is defined as follows:

$$\textbf{Inconsistency IV}(y) := \eta_V > y \tag{4}$$

Both categories can also be combined, if both values fall below the respective thresholds. Similar to the category **Inconsistency I**, it may also be reasonable to assume that two entries with same *ID* in different sections are not related to one another. Hence, every entry which fulfills the conditions for **Inconsistency III** and **Inconsistency IV** or is in a different section as his comparison entry, belongs to its own category **Inconsistency III & IV**. Following the notion of **Inconsistency II** we define **Inconsistency V**. This is the category containing the entries that would be placed using entries which fall into the inconsistency categories of the existing entries.

4.3 Limitations of the Approach

Due to their many advantages graphical models are nowadays used more and more to specify systems [STP12]. Nevertheless, the examination of graphical representations of information was out of scope of our approach. Still the presented approach may for instance be combined with recent research on finding duplicates in UML images [He16]. In addition, although natural language is perceived inferior to more formal graphical notations, there are still a number of reasons why text is needed [BVR17]. Also, the combined use of textual descriptions and graphical representations is considered beneficial to increase understanding [BJM08]. Therefore, we think the proposed approach is a meaningful contribution to improve current as well future specification processes.

5 Evaluation

To evaluate the quality of the resulting document and to get an idea of how to choose the thresholds, we conducted a case study. We designed our case study along the recommendations of Runeson and Höst [RH09]. Our research objective is:

Research Objective: We want to determine suitable thresholds and want to assess the quality of the resulting requirements specification document.

Research Object: To evaluate our approach an industry partner provided us with two specification documents of one system. One document consists of 1,711 entries and the other document consists of 2,552 entries. The former document was created after the latter.

Research Execution: In order to rate the quality of the merged document, we first need to know what thresholds might be suitable. To get a first impression we decided that duplicates must be exactly the same ($\theta_D = \eta_D = 1$) and that unrelated entries are totally different ($\theta_V = \eta_V = 0$). This way, we can assume all entries found as duplicates are definitely duplicates and all entries found as different are in fact unrelated. We use this first result to set the thresholds. The requirements specification with 2,552 entries is added to the requirements specification with 1,711 entries. The rationale is that we expect the smaller document to be in a better condition due to the fact that it was created later.

5.1 Results and Discussion

The first results are shown in Table 1. Considering the sum of all the categories, one can see that all of the 2,552 entries were categorized. Most notably, there is no element in the category **Inconsistency IV**. This is due to the fact that a positional similarity of 0% also means that the sections of entries differ. As a consequence these entries were categorized in **Inconsistency III & IV** instead of **Inconsistency IV**.

Tab. 1: First results

Category	Amount
Unique Element	1,178
Duplicate	392
Variant	635
Inconsistency I	16
Inconsistency II	190
Inconsistency III	45
Inconsistency IV	0
Inconsistency III & IV	82
Inconsistency V	14

Tab. 2: Final result

Category (Existing entries)	Amount
Duplicate	712
Variant	335
Inconsistency III	45
Inconsistency IV	10
Inconsistency III & IV	52
Wrong Inconsistency	7
Inconsistency not detected	2
Category (Unique entries)	**Amount**
Wrongly placed	14
Wrongly included	51
Wrong Inconsistency	1

We used these first results to perform a manual analysis. We wanted to know whether the entries were categorized and placed correctly. The insights were used to set thresholds in a way to minimize wrong categorizations. In addition, we took into account the opinions of stakeholders. They mentioned: every entry that has changed needs to be included, since the system contains safety related requirements which need to be checked manually anyway. This reflects in the threshold for the textual similarity of duplicates being 100%. The resulting thresholds are displayed in equation 5.

$$\theta_D = 100\%,\ \theta_V = 12\%, \eta_D = 20\%,\ \eta_V = 20\% \tag{5}$$

The fact that the positional thresholds are the same contradicts our assumption that the position can be used to separate the categories. Eventually, we merely used them to optimize the result. Whether they are indeed useful, could be a topic for further case studies. Table 2 shows the results using the determined thresholds. Since the thresholds do not have an effect on the added entries, these results have not changed and are not displayed.

In contrast to the first results more entries were classified as duplicates instead of variants, since even minor changes in the position lead to the original categorization as variants. In total, 8 entries were classified as inconsistent although they were not. For these entries our assumption that the section must stay the same did not hold. Additionally, there are 2 entries which are inconsistent but could not be categorized as such using our thresholds. 51 entries were added although they already existed in $RS_{150\%}$. For these entries the *ID* had changed although the text stayed the same. Furthermore, 14 entries were placed in a wrong position. This happened, since the capability of structuring the document was not always used correctly. For instance, entries that should have been child entries were put wrongly on the same level as their parent entry. Hence, wrong entries were used for the placement.

All in all 75 (wrong / not detected inconsistency, wrong placement, wrongly included) of the 2,552 entries were treated in a wrong manner. Inconsistencies between the documents and unforeseen situations are the main reasons for the deficiencies. As a result, the approach should not be used on its own, but as a supportive means to improve the efficiency of the requirements elicitation process.

5.2 Threats to Validity

The thresholds are based on a manual analysis of a requirements specification document containing more than 2000 entries. This task was performed by one person and due to its manual nature is error-prone. Since we used this analysis to set the thresholds in a way to minimize wrong categorizations, the suggested thresholds can only be used as a first idea. The thresholds need to be refined by repeating the evaluation with different and more diverse data. Furthermore, because of the ambiguous nature of natural language, there will be no gold-standard – even with more refined thresholds.

6 Conclusion and Outlook

In this paper we proposed an approach to automatically merge requirements specification documents of multiple product variants into a single document. The purpose of the merged document is to facilitate requirements reuse and hence reduce errors during requirements elicitation. We explain what preconditions must be fulfilled to adopt the approach. The approach itself compares the entries of documents and decides how to incorporate additional information. To avoid deficiencies in the resulting document, we introduced a number of inconsistencies. The evaluation shows that the approach is applicable to real requirements specification documents. Out of 2,552 regarded entries 75 entries are treated incorrectly.

Since the parameters we determined are based on two documents of one system, they are not applicable in general. Hence the evaluation needs to be repeated to gain more general thresholds. Aside from the merge process itself, it showed that the manual assignment of the

identifiers is a major source for inconsistencies between the documents. The application of an automatic approach to assign the identifiers may reduce inconsistencies and save effort in the creation of the documents.

References

[Al10] Alves, Vander; Niu, Nan; Alves, Carina; Valença, George: Requirements engineering for software product lines: A systematic literature review. Information and Software Technology, 52(8), 2010.

[ALL12] Apel, Sven; Leßenich, Olaf; Lengauer, Christian: Structured Merge with Auto-tuning: Balancing Precision and Performance. In: 27th IEEE/ACM International Conference on Automated Software Engineering. 2012.

[BJM08] Burton-Jones, Andrew; Meso, Peter N: The Effects of Decomposition Quality and Multiple Forms of Information on Novices' Understanding of a Domain from a Conceptual Model. Journal of the Association for Information Systems, 9(12), 2008.

[BKK03] Berry, DM; Kamsties, E; Krieger, MM: From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity-A Handbook . U. Waterloo, 2003.

[Br06] Brunet, Greg; Chechik, Marsha; Easterbrook, Steve; Nejati, Shiva; Niu, Nan; Sabetzadeh, Mehrdad: A Manifesto for Model Merging. In: International Workshop on Global Integrated Model Management. 2006.

[BVR17] Beckmann, Martin; Vogelsang, Andreas; Reuter, Christian: A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents. In: 25th IEEE International Requirements Engineering Conference. 2017.

[Ch12] Chernak, Yuri: Requirements Reuse: The State of the Practice. In: IEEE International Conference on Technology and Engineering (SWSTE). 2012.

[Da01] och Dag, J Natt; Regnell, Björn; Carlshamre, Pär; Andersson, Michael; Karlsson, Joachim: Evaluating Automated Support for Requirements Similarity Analysis in Market-Driven Development. In: 7th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'01). 2001.

[EBB09] Eriksson, Magnus; Börstler, Jürgen; Borg, Kjell: Managing requirements specifications for product lines–An approach and industry case study. Systems and Software, 82(3), 2009.

[FCC13] Falessi, Davide; Cantone, Giovanni; Canfora, Gerardo: Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques. IEEE Transactions on Software Engineering, 39(1), 2013.

[FE00] Finkelstein, Anthony; Emmerich, Wolfgang: The Future of Requirements Management Tools. In: Information Systems in Public Administration and Law. Österreichische Computer Gesellschaft, 2000.

[GB15] Goldin, Leah; Berry, Daniel M: Reuse of requirements reduced time to market at one industrial shop: a case study. Requirements Engineering, 20(1), 2015.

[Gr14] Grönninger, Hans; Hartmann, Jochen; Krahn, Holger; Kriebel, Stefan; Rothhart, Lutz; Rumpe, Bernhard: Modelling Automotive Function Nets with Views for Features, Variants, and Modes. In: International Conference on Technology and Engineering (SWSTE). 2014.

[He16] Hebig, Regina; Quang, Truong Ho; Chaudron, Michel RV; Robles, Gregorio; Fernandez, Miguel Angel: The Quest for Open Source Projects that use UML: Mining GitHub. In: 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. 2016.

[Ka15] Kassab, Mohamad: The Changing Landscape of Requirements Engineering Practices over the Past Decade. In: 5th IEEE International Workshop on Empirical Requirements Engineering. 2015.

[Kn02] von Knethen, Antje; Paech, Barbara; Kiedaisch, Friedemann; Houdek, Frank: Systematic Requirements Recycling through Abstraction and Traceability. In: Joint International Conference on Requirements Engineering. 2002.

[LJB98] Lam, Wing; Jones, Sara; Britton, Carol: Technology Transfer for Reuse A Management Model and Process Improvement Framework. In: 3rd IEEE International Conference on Requirements Engineering. 1998.

[LMV97] Lam, Wing; McDermid, John A; Vickers, AJ: Ten Steps Towards Systematic Requirements Reuse. Requirements Engineering, 2(2), 1997.

[Me02] Mens, Tom: A State-of-the-Art Survey on Software Merging. IEEE Transactions on Software Engineering, 28(5), 2002.

[PFQ14] Palomares, Cristina; Franch, Xavier; Quer, Carme: Requirements Reuse and Patterns: A Survey. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, 2014.

[RCC13] Rubin, Julia; Czarnecki, Krzysztof; Chechik, Marsha: Managing Cloned Variants: A Framework and Experience. In: Proceedings of the 17th International Software Product Line Conference. ACM, 2013.

[RH09] Runeson, Per; Höst, Martin: Guidelines for conducting and reporting case study research in software engineering. volume 14. Springer, 2009.

[So12] Sommerville, Ian: Software Engineering. Addison-Wesley, USA, 9th edition, 2012.

[STP12] Sikora, Ernst; Tenbergen, Bastian; Pohl, Klaus: Industry needs and research directions in requirements engineering for embedded systems. Requirements Engineering, 17(1), 2012.

[TC12] Tekli, Joe; Chbeir, Richard: A novel XML document structure comparison framework based-on sub-tree commonalities and label semantics. Web Semantics: Science, Services and Agents on the World Wide Web, 11, 2012.

[Th11] The Institute of Electrical and Electronics Engineers, Inc.: ISO 29148:2011, Systems and Software Engineering – Life cycle processes –Requirements Engineering. 2011.

[We91] Westfechtel, Bernhard: Structure-Oriented Merging of Revisions of Software Documents. In: International Workshop on Software Configuration Management. ACM, 1991.