A Semi-automated Approach for Generating Natural Language Requirements Documents based on Business Process Models

Article *in* Information and Software Technology · January 2018

DOI: 10.1016/j.infsof.2017.08.009

CITATIONS
53

READS
797

4 authors, including:

Banu Aysolmaz
Eindhoven University of Technology
58 PUBLICATIONS **506** CITATIONS
SEE PROFILE

Henrik Leopold
Kühne Logistics University
123 PUBLICATIONS **3,643** CITATIONS
SEE PROFILE

Onur Demirors
Izmir Institute of Technology
264 PUBLICATIONS **2,660** CITATIONS
SEE PROFILE

A Semi-automated Approach for Generating Natural Language Requirements Documents based on Business Process Models

Banu Aysolmaz[a,b,*], Henrik Leopold[a], Hajo A. Reijers[a], Onur Demirörs[c,d]

[a] *Vrije Universiteit Amsterdam, Department of Computer Science, De Boelelaan 1105, 1081HV Amsterdam, The Netherlands*
[b] *Maastricht University, School of Business and Economics, PO Box 616, 6200 MD, Maastricht, the Netherlands*
[c] *Izmir Institute of Technology, Department of Computer Engineering, 35430, Urla, Turkey*
[d] *University of New South Wales, School of Computer Science and Engineering, Barker St, Kensington NSW 2052, Australia*

Abstract

Context: The analysis of requirements for business-related software systems is often supported by using business process models. However, the final requirements are typically still specified in natural language. This means that the knowledge captured in process models must be consistently transferred to the specified requirements. Possible inconsistencies between process models and requirements represent a serious threat for the successful development of the software system and may require the repetition of process analysis activities.

Objective: The objective of this paper is to address the problem of inconsistency between process models and natural language requirements in the context of software development.

Method: We define a semi-automated approach that consists of a process model-based procedure for capturing execution-related data in requirements models and an algorithm that takes these models as input for generating natural language requirements. We evaluated our approach in the context of a multiple case study with three organizations and a total of 13 software development projects.

Results: We found that our approach can successfully generate well-readable requirements, which do not only positively contribute to consistency, but also to the completeness and maintainability of requirements. The practical use of our approach to identify a suitable subcontractor on the market in 11 of the 13 projects further highlights the practical value of our approach.

Conclusion: Our approach provides a structured way to obtain high-quality requirements documents from process models and to maintain textual and visual representations of requirements in a consistent way.

Keywords: Requirements elicitation, business process model, natural language generation

[0]DOI and link to published article: https://doi.org/10.1016/j.infsof.2017.08.009
*Corresponding author
Email addresses: b.aysolmaz@maastrichtuniversity.nl,banuays@gmail.com (Banu Aysolmaz), h.leopold@vu.nl (Henrik Leopold), h.a.reijers@vu.nl (Hajo A. Reijers), onurdemirors@iyte.edu.tr (Onur Demirörs)

1. Introduction

Business process modeling is an established method for documenting, analyzing, and improving organizational operations. What is more, it has become a widely accepted practice in software engineering [1, 2, 3]. In particular for analyzing requirements of business-related software systems business process modeling has proven to be an effective means [4]. Process models do not only provide an overview of the operations that must be supported by the to-be developed software systems, but also show how these operations are related to the different organizational roles and systems.

Despite this prominent role of business process modeling for requirements analysis, the actual specification of requirements is commonly conducted using natural language [5, 6, 7, 8]. This means that the knowledge captured in process models must be consistently transferred to natural language requirements. On the one hand, this is a complex and time-consuming task [9, 10]. On the other hand, updates at later stages in either the textual or the model-based requirements come with the risk of inconsistencies [11, 12, 13]. Such inconsistencies between the process model and the resulting requirements represent a serious threat for the successful development of the respective software system throughout the Software Development Lifecycle (SDLC). More specifically, they may result in a system that does not fully reflect the functionality defined in the process models.

To address this problem, we propose a semi-automated approach whose final output are generated requirements documents that integrate process model and execution-related data in an understandable fashion. As a result, organizations can systematically transfer the knowledge captured in their process models to other SDLC activities and create consistent and maintainable artifacts. Our proposed approach consists of three main steps. In the first step, we analyze the process models that are relevant for the system to be developed and identify the set of automatable activities. In the second step, we capture execution-related data, such as responsibilities, application systems, data needs, and additional constraints in a requirements model. In the third step, we automatically generate requirements documents from the created models via a template-based natural language generation algorithm. The consistency between the processes and the requirements is by definition guaranteed by the generation feature of the approach. To evaluate the impact of our approach on other key characteristics of high-quality requirements –readability, completeness and maintainability– , we conducted a multiple case study that involved 3 different organizations and a total of 13 software development projects. We found that the requirements documents generated by our approach were considered to be well-readable, almost perfectly complete, and beneficial for improving consistency as well as maintainability. Meeting these key requirement characteristics was found to be essential to enhance the usability of the requirements by domain experts, analysts, project managers, and software developers. In 11 of the projects, the generated artifacts were used for identifying a suitable subcontractor on the market for developing the respective systems, which confirmed the usability of the approach in practical settings.

The remainder of this paper is structured as follows. In Section 2 we elaborate on the background of our research and identify the research gap that we will address. In Section 3, we introduce our semi-automatic approach for generating requirements documents. In Section 4, we present and discuss the findings of our multiple case study. In Section 5, we elaborate on the steps required for adapting the presented approach to languages other than English. In Section 6 we discuss the implications of our work before concluding the paper in Section 7.

2. Background

In this section, we discuss the background of our paper. In Section 2.1, we first clarify the relevance and the value of generating natural language requirements. In Section 2.2, we then elaborate on the use of process models in requirements engineering. We close the section by pointing out what is still missing to define an approach for automatically generating high quality requirements from process models.

2.1. The Value of Requirements Generation

While many would argue that models are the preferred means to foster communication, others favor requirements in textual format. At its heart, the question about the value of generating natural language requirements relates to the debate whether textual or visual representations are superior in terms of communication effectiveness. Interestingly, this debate is neither new nor limited to the field of requirements engineering. The first studies addressing this controversy date back to the seventies. At this time, psychologists empirically compared the expressive power of natural language texts with matrices, spatial maps, and tree representations [14, 15, 16, 17]. Later, many studies from the field of computer science contributed to the debate. Among others, authors compared the comprehension performance of code-based representations and flow diagrams [18, 19, 20]. The conclusions of these and other works remain, however, contradictory. Some argue in favor of text-based other argue in favor of visual representations.

A satisfying explanation for these opposing views is provided by the Cognitive Theory of Multimedia Learning (CTML) [21], which has been developed through more than a decade of empirical research. Among others, it discusses the concept of *learning preference*, which suggests that both textual and visual representations should be presented at the same time. The rationale behind this concept is that people with different backgrounds may simply have different preferences and cognitive abilities. By providing both representations, they are provided with a choice.

Transferred to the field of requirements engineering, the CTML suggests that both models and natural language requirements should be used for capturing and discussing requirements. In fact, this view is supported by many researchers. For instance, Weber and Weisbrod discuss the importance of natural language requirements for communication, but also highlight that the sole use of natural language is hardly

Table 1: Work combining process models and requirements engineering

Approach	Authors
Manual use of process models	
Elicitation of textual requirements	
Requirements engineering based on business process models	Cardoso et al. [4]
Business process modeling and requirements modeling	Mayr et al. [28]
Process-oriented information system requirements engineering	Ma and Jiang [7]
A business process-driven approach for requirements dependency analysis	Li et al. [8]
Utilizing business process models for requirements elicitation	Demirörs et al. [29]
Requirements elicitation using BPM notations	Monsalve et al. [30]
Elicitation of model-based requirements	
A goal-based approach on business process-driven requirements engineering	González and Díaz [31]
Deriving requirements from process models via the problem frames approach	Cox et al. [32]
Automated use of process models	
Transformation of business process models to business rules	Malik and Bajwa [33]
Supporting process model validation through natural language generation	Leopold et al. [9]
Generating functional requirements from process models	Türetken et al. [34]
Bridging the gap between business process modeling and software requirements analysis	Coşkunçay et al. [35]

feasible for complex projects [22]. They propose the additional use of so-called requirements management information models (RMIs). In a similar way, Schatz et al. [23] and Davis [24] propose to combine text-based and model-based requirements. Nicolás and Toval even explicitly discuss the value of generation in this context [25]. They argue that generation reduces the effort and, at the same time, increases the quality and traceability of the requirements.

Recognizing the potential of automatically generating natural language requirements, we define a respective approach for process models in this paper. To highlight what is specifically missing to define such an approach, the next section reviews related work on process models in the context of requirements engineering.

2.2. Process Models and Requirements Engineering

Many authors have emphasized the important role of process models in the context of specifying requirements of software systems [26, 27, 28]. Some authors even go so far as considering their use as mandatory [1, 3]. However, the specific role of process models differs considerably among available approaches. Table 1 gives an overview of the most relevant works using process models in the context of requirements engineering. As Table 1 illustrates, we differentiate between works that use process models in a manual and in an automated way.

The related work that discusses *the manual use of process models* in the context of requirements engineering can be further categorized into works that elicit *textual* and that elicit *model-based* requirements from process models.

The main insight of the works from the first subcategory that elicit *textual* requirements from process models is that process models represent an effective way of steering the activity of requirements elicitation

and enhance the completeness, correctness, and traceability of the final requirement statements [4]. Cardoso et al. analyze the level of automation for each activity in the process models and then define a set of textual requirements for the activities to be automated [4]. In a similar manner, Ma and Jiang define a set of textual requirements for each activity of a process [7]. Mayr et al. discuss that detailed notions for requirements should be specified based on process models and they also map requirements in sentence form to the process models [28]. Li et al. propose a method to link textual requirements to activities in the process model [8]. Such links help to identify dependencies between requirements consecutively being used for discovering missing and ambiguous text-based requirements. Demirörs et al. analyze and define not only functional requirements, but also non-functional, security, and hardware requirements based on process models. Lastly, Monsalve et al. elaborate on the usage of process modeling notations for eliciting and expressing user requirements on a strategic level. They find Qualigram more helpful in this respect than BPMN [30]. What all these works have in common is that they exemplify how process models can support requirements elicitation. What is more, they show that process models are also useful for identifying gaps and problems, thus for validating requirements with end users.

The second subcategory of works that elicit *model-based* requirements from process models illustrates that process models are also useful for deriving model-based requirements. For instance, González and Díaz suggest to build a goal model using the activities from process models [31]. They subsequently use the goal model to establish the use cases and their relations. However, the specification remains on the use case diagram level and the usage of the suggested role and resource models in the context of the requirements definition is left open. Cox et al. discuss that the framing of real-world problems for capturing and classifying software development problems is a difficult task in reality. They define a set of steps to manually develop problem frame diagrams together with textual requirements using role activity diagrams. Rather than being an elicitation and validation tool between domain experts and modelers, the problem frames approach enables the formal analysis of requirements for verification. What both approaches have in common is that they enhance the representational capabilities of process models for requirements elicitation. However, they do not consider automated support.

Related work on *the automated use of process models* in the context of requirements engineering consider process models as the final requirements artifact and focus on the benefits of verbalizing the models in the requirements elicitation and validation phases. For instance, Leopold et al. analyze the activity labels and the control flow of process models to automatically generate corresponding natural language descriptions of the models [9]. Malik and Bajwa provide a sentence generation algorithm for requirements using a template-based approach [33]. Though their approach does not include clear text structuring techniques, the consideration of the message flow between parties is an important feature to reveal requirements on system interactions. Türetken et al. include a broader set of process elements in the generated sentences, including roles, input and output data, events, and systems [34]. Consideration of such elements is important to be

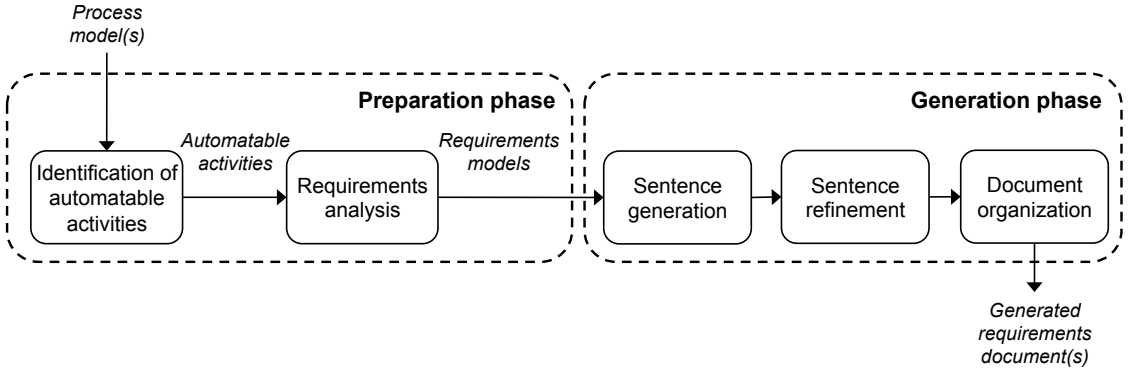


Figure 1: Overview of our approach

able to express requirements that concern other aspects than control flow. However, they rely on a certain process structure, do not consider all execution-related aspects, and only generate rudimentary sentences. The work of Coşkunçay et al. specifies the need for analyzing additional data for process automation in a separate set of models, though it lacks a description of requirements analysis approach and a formal generation technique. The studies in this group commonly express the need for the automation of natural language requirement specification based on process models.

This literature review showed that process models play an important role in the context of analyzing and representing system requirements. What is more, it showed that first approaches considering the automated generation of requirements based on process models have already been introduced. What is still missing is an approach that integrates the complete set of execution-related data and provides the user with consistent, well-readable, and also well-maintainable requirements. Recognizing this gap, we use this paper to propose a semi-automated approach that automatically generates textual requirements documents based on process models and execution-related data. We will show that our approach provides a structured way to obtain consistent requirements that are well readable, complete, and easy to maintain.

3. Conceptual Approach

In this section, we introduce our approach for the semi-automated generation of requirements documents based on process models. As illustrated by Figure 1, the approach consists of two main phases: a preparation phase and a generation phase. In the preparation phase, we first analyze the input process model(s) and identify automatable activities. Then, we analyze the requirements for the automated execution of these activities and create a requirements model for each of them. In the generation phase, these requirements models are used as the input for the automated generation of the requirements documents. In the following subsections, we introduce the details of each phase and illustrate our concepts using a running example.

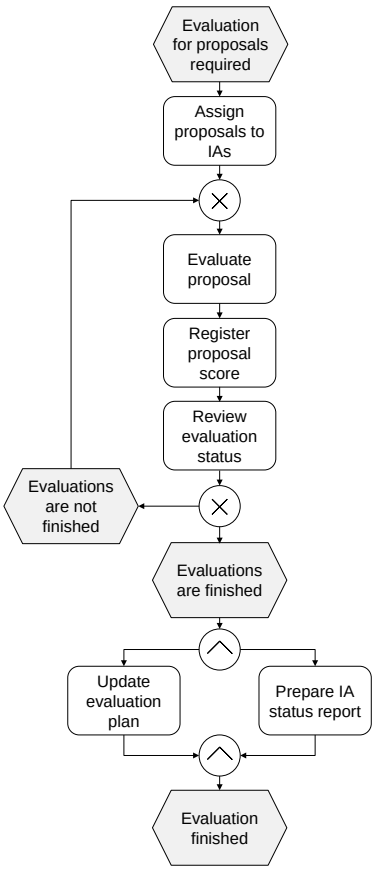


Figure 2: Exemplary EPC Model for Daily Independent Auditor Evaluation Process

3.1. Preparation Phase

The starting point of the preparation phase is a set of process models. We manually analyze each of the input process models to *identify automatable activities*. Activities that can either be supported by the system to be developed or can be totally automated are marked and added to the list of automatable activities. Unclear cases are discussed with the respective process owners. The result of this step is a set of automatable activities that constitute the basis for associating the underlying business processes with the requirements.

To illustrate this step, consider the business process shown in Figure 2. It describes the evaluation of project proposals by independent auditors (IAs) in the context of a grant program. It is depicted using the Event Driven Process Chain (EPC) notation, a modeling language widely used in industry [36]. In this paper, we use the EPC notation as an example to illustrate our approach. Note, however, that our approach can be applied to other process modeling notations such as the Business Process Model and Notation (BPMN) without adaptations. The example process from Figure 2 is triggered when evaluations for proposals are required. The first activity is to assign the proposals to IAs. Once the proposals have been assigned to IAs,

they are evaluated. Then, the proposal score is registered and the evaluation status is reviewed. In case the evaluations are not yet finished, they are evaluated by other IAs. Otherwise, the evaluation plan is updated and a status report is prepared. Upon closer inspection of the process model from Figure 2, it becomes clear that it contains four automation candidates: "*Assign proposals to IAs*", "*Register proposal score*", "*Update evaluation plan*", and "*Prepare IA status report*". The other activities must be performed manually and are outside the scope of the system to be developed. These activities are "*Evaluate proposal*" and "*Review evaluation status*".

The second step of the preparation phase is the *requirements analysis*. The main goal of this step is to specify how the activities are to be executed. This requires the identification of execution-related data for activities. Building on the insights from [3, 37, 38, 39], we investigate the following four execution-related aspect for each automatable activity:

- *Responsibilities*: To specify the responsibilities associated with an activity, we adopt the so-called RASCI matrix [40, 41]. This means that we do not only capture the different roles that are involved in the execution of the activity, but also capture their specific responsibilities, such as "*carries out*" or "*approves*". In conformance with the RASCI concept [42], we also capture whether multiple roles share the specified responsibility (e.g. whether multiple roles may "*carry out*" or "*approve*" the activity) or whether the role has the exclusive responsibility (e.g. only that role can "*carry out*" or "*approve*" the activity).

- *Data needs*: As for the data needs, we specify how data entities are used by the activity [43, 44]. Therefore, we adopt the *CRUDL* approach and capture manipulation operations (create, update, delete) and usage operations (read, use, view, list).

- *System interactions*: During the execution of an activity, interactions with multiple systems may take place. We identify both internal applications that are to be developed as part of the system and external applications that the system communicates with (e.g., web services). In this way, not only internal entity operations, but also data interface requirements are revealed.

- *Execution constraints*: In addition to the later three aspects, we also capture *constraints* of the application system during the execution of the considered activity. As categorized by Goedertier and Vanthienen, possible business constraints can, for instance, emerge from business regulations, business policies, costs and benefits, time, information prerequisites, and technical circumstances [45].

Typically, the information about these aspects must be obtained from domain experts who are part of the respective business processes. We propose the use of the following questions to infer the required information:

- (Q1) Who will be responsible to perform this activity and what will be the responsibility types involved?
- (Q2) What are the data entities needed to execute this activity and how are they used?
- (Q3) Which internal and external systems are interacted with for the execution of this activity?
- (Q4) What constraints and rules need to be taken into account during the execution of this activity?

Based on these questions, we elicit the relevant functional requirements from the domain experts and capture the results for each activity in a requirements model. More specifically, we use a customized version of the so-called Function Allocation Diagram (FAD) introduced as part of the ARIS method [36]. FADs are used to focus on the details of an individual activity by depicting the process elements related to that activity. For complete requirements, we need to represent the aforementioned four execution-related aspects in the requirements model. The FAD is a conceptual model that allows us to do so by adding respective model elements for the execution-related aspects. Figure 3 shows an exemplary FAD for the activity "*Register proposal score*". It shows that the activity is associated with three roles. The "*Project Officer*" and the "*Evaluation Committee Member*" are responsible for carrying it out while the "*Independent Auditor*" is responsible for its approval. Note that the marker "+" indicates that a responsibility can be exercised by either of the associated roles. A responsibility without a marker, therefore, represents a responsibility that is jointly exercised by all associated roles. The FAD also specifies the data needs of the activity. Among others, we can see that the "*Project proposal*" is read and the "*Proposal status*" is viewed and updated. We can also see the two systems that are relevant for the activity –the "*Grant Management System* and the "*IA Registration System*"– and how they are connected with the data needs and operations. Lastly, we observe two constraints that are associated with the two systems. They are expressed using natural language and specify that (1) a third evaluation is requested in case two evaluations differ to a certain degree and that (2) IAs might be dropped if they continuously submit contradicting evaluations.

In the next section, we explain how such a requirements model can be used for the automated generation of a requirements document.

3.2. Requirements Document Generation

This section defines our approach for generating textual requirements documents from the requirements models defined in the preparation phase. In line with other natural language generation systems, we adopt the traditional pipeline concept [46]. In particular, as outlined by Figure 1, we follow a three step procedure. First, we generate the sentences from the requirements models. Then, we refine the generated sentences by aggregating them in a way that appeals to the user. Finally, we organize the generated sentences in the context of a document structure. In the sub sections 3.2.1 through 3.2.3, we explain the details of each step.

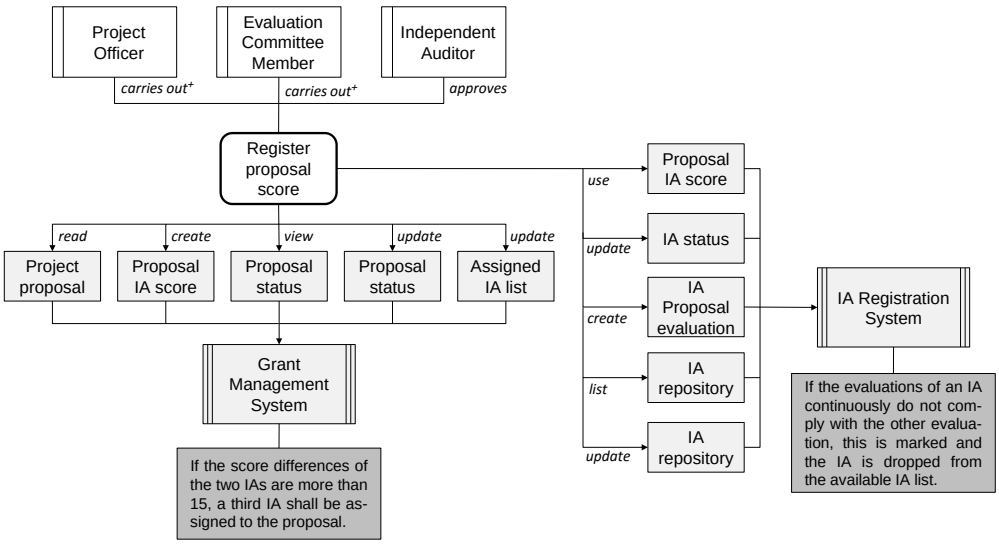


Figure 3: The FAD for *Register proposal score* activity in "Daily IA Evaluation" process.

3.2.1. Sentence Generation

To adequately reflect the information captured in the requirements model, we generate three types of sentences. First, we generate sentences describing which roles are involved in the execution of the activity (responsibility sentences). Second, we generate sentences specifying the usage and manipulation of the data entities (data need sentences). Third, we generate sentences describing the constraints (constraint sentences). Note that there is no dedicated sentence type for *system interactions*. They are either covered by data need sentences (if the system interaction relates to a data need) or constraint sentences.

To implement the generation of these different sentence types, we adopt the so-called template filling approach [47]. The rationale behind this approach is to define sentence templates which contain well-defined gaps. By filling a template with the respective information (in our case the information from a requirements model), proper sentences are constructed in an automated fashion. The advantages of such template filling approaches are their speed, the consistency of the produced sentences, and the high linguistic quality of the output. What is more, it does not require any specific knowledge related to natural language generation to adapt the system [47]. Hence, they are often considered as a viable choice for natural language generation [48]. Table 2 gives an overview of the sentence templates we defined for the three sentence types. The first three templates (R1 to R3) are used to generate sentences about the responsibilities associated with the activity and, therefore, answer question Q1. The templates on data needs (D1 and D2) serve the purpose of generating sentences with respect to questions Q2 and Q3. Lastly, the answer to the question Q4 is provided by means of the sentences generated by template C1. The gaps in the templates that need to be filled with information from the requirements model are indicated by terms between "<" and ">". While the terms for roles, responsibilities, entities, operations, and systems are directly obtained from the

Table 2: Sentence templates for requirements generation

Type	No.	Sentence template
Responsibility	R1	The <*Role*> shall <*Action*> <*Object*>.
	R2	The <*Role*> shall <*Responsibility*> the operation of <$Action_{Gerund}$> <*Object*>.
	R3	The <*System*> shall automatically <*Action*> <*Object*>.
Data need	D1	While <$Action_{Gerund}$> and by using the <*Entity*>, operations shall be performed on the <*System*>.
	D2	While <$Action_{Gerund}$>, the <*Entity*> shall be <$Operation_{Participle}$> on/from the <*System*>.
Constraint	C1	<$Constraint_{Condition}$> while <$Action_{Gerund}$> on the <*System*>, <$Constraint_{Consequence}$>.

labels of the model, the activity is split into an action (i.e., the verb) and an object, and the constraint is split into a condition and a consequence. Both operations can be automatically performed using available tools. Deriving action and object from activity labels is possible with the technique introduced in [49] and splitting conditional sentences can be implemented using the Stanford Parser [50]. Note that verbs may occur in different grammatical forms (i.e., base form, gerund, and participle).

Algorithm 1 formalizes the steps of our template-based sentence generation approach. The algorithm requires a requirements model (e.g. an FAD) as input. As a result, it returns a list of sentences.

The algorithm starts with the creation of a list s for the generated sentences (line 1). The first part of the algorithm is then concerned with generating the responsibility sentences (lines 2-17). It begins by checking whether the considered requirements model contains roles (line 2). If that is the case, it is checked whether all roles exclusively perform "*carry out*"-operations (line 3). If yes, a responsibility sentence using template R1 is created for each role (lines 4-7). To this end, the required information (role, action, and object) are derived from the requirements model. If the roles also perform other operations, a responsibility sentence using template R2 is created for each role (lines 9-12). Since this template requires the action in the gerund form (e.g. "*defining*" instead of "*define*"), we use the lexical database WordNet to derive the gerund form from the base form. In case the considered responsibility model does not contain any roles, a responsibility sentence using template R3 is created (lines 15-16). Instead of using a role description, this sentence uses the name of the main system.

The second part of the algorithm handles the generation of the data need and constraint sentences (lines 18-34). For this purpose each system from the requirements model is analyzed separately. For each system, the algorithm then analyzes the respective operations that are associated with this system (lines 19-27). If a considered operation is of type "*use*", a sentence using template D1 is created (lines 20-22). For other operations than "*use*", a sentence using template D2 is created (lines 23-25). This requires the placement of the participle form of that operation name. Finally, respective sentences for the constraints are

Algorithm 1: generateSentences(RequirementsModel rm)

```
1: List sentences = new List();
2: if rm.getRoles() ≠ ∅ then
3:    if rm.getResponsibilities().containsOnly("carry out") = true then
4:       for all Role r ∈ rm.getRoles() do
5:          Sentence s = fillTemplateR1(r,rm.getAction(),rm.getObject());
6:          sentences.add(s);
7:       end for
8:    else
9:       for all Role r ∈ rm.getRoles() do
10:         Sentence s = fillTemplateR2(r,transformToGerund(rm.getAction()),rm.getObject());
11:         sentences.add(s);
12:      end for
13:   end if
14: else
15:   Sentence s = fillTemplateR3(rm.getMainSystem(),rm.getAction(),rm.getObject());
16:   sentences.add(s);
17: end if
18: for all System sys ∈ rm.getSystems() do
19:   for all Operation o ∈ sys.getOperations() do
20:      if o = "use" then
21:         Sentence s = fillTemplateD1(o.getEntity(),sys,transformToParticiple(rm.getAction()));
22:         sentences.add(s);
23:      else
24:         Sentence s = fillTemplateD2(o.getEntity(),sys,transformToGerund(rm.getAction()));
25:         sentences.add(s);
26:      end if
27:   end for
28:   if rm.getConstraints() ≠ ∅ then
29:      for all Constraint c ∈ sys.getConstraints() do
30:         Sentence s = fillTemplateC1(c.getCondition(),c.getConsequence(),sys,transformToGerund(rm.getAction()));
31:         sentences.add(s);
32:      end for
33:   end if
34: end for
35: return s;
```

generated (lines 28-33). In case the considered system is associated with one or more constraints, a sentence following the template C1 is created for each of these (lines 30-31). Once all available components from the requirements model are verbalized, the list of sentences s is returned and the algorithm has completed.

To illustrate the effect of Algorithm 1, Table 3 provides examples for sentences generated based on the requirements model from Figure 3.

3.2.2. Sentence Refinement

In this step, we refine the generated responsibility and data need sentences to enhance their readability. We apply one aggregation technique for responsibility sentences and three aggregation techniques for data need sentences as described below.

- *Role aggregation*: If the same responsibility type is applicable for multiple roles, we merge the respective sentences. For instance, instead of keeping the two sentences "*The Project Officer shall register the*

Table 3: Exemplary sentences generated from the requirements model from Figure 3

Type	**No.**	**Example**
Responsibility	R1	The Project Officer shall carry out the operation of registering the proposal score.
	R2	The Independent Auditor shall approve the operation of registering the proposal score.
Data need	D1	While registering the proposal score, the project proposal shall be read from the Grant Management System.
	D2	While registering the proposal score and by using the proposal IA score, operations shall be performed on the IA Registration System.
Constraint	C1	If the score differences of the two IAs are more than 15 while registering the proposal score on the Grant Management System, a third IA shall be assigned to the proposal.

proposal score" and "*The Committee Member shall register the proposal score*", we generate the refined sentence "*The Project Officer or the Committee shall register the proposal score*". Note that depending on the specific connection (as indicated by the presence of a marker), we insert the correct conjunction to express a shared or exclusive responsibility among multiple roles. In case of a marker, we respectively insert the conjunction "*or*", otherwise we insert the conjunction "*and*". If more than one responsibility type is used, the sentences are combined to include those types. For instance, we generate "*The Project Officer shall register, and the Independent Auditor shall approve the operation of registering the project score*".

- *Object aggregation*: If the model contains multiple entities with the same operations, we merge the sentences. For instance, instead of keeping "*While registering the proposal score, the proposal status shall be updated on the Grant Management System*" and "*While registering the proposal score, the assigned IA list shall be updated on the Grant Management System*", we generate the refined sentence "*While registering the proposal score, the proposal status and the assigned IA list shall be updated on the Grant Management System*".

- *Operation aggregation*: If the model contains multiple operations for the same entity, we apply the same procedure as for the object aggregation. For instance, we generate "*the IA repository shall be listed and updated*" instead of discussing these aspects in different sentences.

- *System aggregation*: If the model contains multiple systems, we further merge the previously refined sentences into a single one. For instance, we merge the sentences for two systems as "*While registering the proposal score, the project proposal shall be read on the Grant Management System, and the proposal evaluation shall be created on the IA Registration System*" instead of having two different sentences for each system. This is an optional refinement step and it is only applied when the requirements model

includes a small number of entities.

As a result of applying four aggregation techniques on the sentences generated in the example requirements model in Figure 3, the following refined sentences are obtained:

1. "*The Project Officer or the Evaluation Committee Member shall carry out, and the Independent Auditor shall approve the operation of registering the proposal score.*"
2. "*While registering the proposal score, the project proposal shall be read, the proposal status shall be viewed and updated, the proposal IA score shall be created, and the assigned IA list shall be updated on the Grant Management System.*"
3. "*While registering the proposal score and by using the proposal IA score, the IA repository shall be listed and updated, the IA proposal evaluation shall be created, and the IA status shall be updated on the IA Registration System.*"

3.2.3. Structuring of the Document

Upon completion of the two phases of the generation, the requirements sentences need to be organized in the context of a document. For this purpose, we assign unique IDs to the aggregated requirements sentences. The requirements document can then be organized in two different ways as explained below.

Process-based document. In this way of organizing the document, we exploit the hierarchy of the considered process models to structure the document. Thus, the application of this type requires a hierarchical organization of the process models. Table 4 illustrates the resulting document structure. The first level heading is derived from the process model on top of the process hierarchy. Then, the requirements sentences generated for the top-level process model are listed. Afterwards, we create a subheading for each of the process models from the levels underneath and list the requirements under the respective subheading. All sentences are organized in the same way by recursively processing all models. The order of the requirements for a specific process is derived from the order of the respective activities.

System-based document. In this document style, we use the systems to be developed to organize the requirements sentences. That is, we list the requirement sentences under the respective headings of the systems. Responsibility sentences are placed under the heading for the main system. In case this style is used, the *system aggregation* technique is not applied on data need sentences in the sentence refinement phase.

Once all these steps have been completed, users are provided with automatically generated requirements documents, organized in their preferred style. In the next section, we apply our semi-automated approach

Table 4: Process-based requirements document structure

1 <Top-level process name> Process
$REQ1_1$ <Responsibility sentence for activity 1>
$REQ2_1$ <Data need sentence for activity 1 and system 1>
$REQ3_1$ <Data need sentence for activity 1 and system 2>
... (all data need sentences)
$REQX_1$ <Constraint sentence for activity 1>
... (all constraint sentences)
$REQ1_N$ <Responsibility sentence for activity N>
$REQ2_N$ <Data need sentence for activity N and system 1>
$REQ3_N$ <Data need sentence for activity N and system 2>
.. (all data need sentences)
$REQX_N$ <Constraint sentence for activity N>
... (all constraint sentences)
1.1 <First-level process name> Process
$REQ1_M$ <Responsibility sentence for activity M>
$REQ2_M$ <Data need sentence for activity M and system 1>
...

in the context of a case study to demonstrate the improvements in obtaining higher-quality requirements in terms of key requirement characteristics.

4. Evaluation

In order to show the feasibility of our approach in practice, we applied it in a real-world setting [51]. More specifically, we conducted a multiple case study using a set of three different organizations and 13 projects with varying characteristics [52]. In this way, we were able to improve the generalizability of the findings and to demonstrate the value of our approach [53]. The overall goal of the evaluation is to learn whether the project teams from our case study perceive the generated requirements documents as well-readable, complete, consistent, and easy to maintain. In Section 4.1, we explain the rationale for the selection of the cases and introduce them in detail. In Section 4.2, we briefly describe our implementation of the approach in the context of a prototype tool. In Section 4.3, we provide details on how we conducted the case study. In Section 4.4, we present the findings of our case study. In Section 4.5, we compare the manually created and generated requirements documents. In Section 4.6, we discuss the limitations of our evaluation.

4.1. Overview of Cases

Process modeling and requirements analysis activities are performed in a wide spectrum of industry fields. This diversity required us to get involved in cases with varying characteristics. While assembling a suitable set of cases, we considered two main goals. First, we wanted to show the value of our approach in a practical setting. Thus, we required one or more organizations that were willing to perform business

Table 5: Overview of the case set characteristics

Case set	Case type	Project	#PM	#ACT
(1) e-Government	New project	e-Company	18	125
		e-Trademark	9	47
		Total	***27***	***172***
(2) Public Services	New project	Auditing	4	53
		Budget Management.	69	470
		Archive Management	12	638
		Human Resource Management	24	218
		Investment	20	123
		Performance Management	6	26
		Program Management	8	953
		Project Support	91	662
		Stakeholder Management	18	729
		Total	***252***	***1782***
(3) Campus System	Retrospective	Announcement	3	25
		Research Program Management	13	42
		Total	***16***	***67***

Legend: #PM = Number of process models, #ACT = Number of activities

process and requirements analysis as part of a system development project. Second, we wanted to compare our approach to traditional requirements engineering. Therefore, we sought at least one organization that was interested in applying our approach after the actual development project, allowing us to retrospectively compare the results.

As a result of these considerations, we selected three *case sets*. Each of these case sets consisted of a program covering multiple projects. Projects in a program were managed by the same integrator organization, which used similar principles and practices. The integrator organization was in charge of subcontracting the projects. Altogether, the case sets included 13 projects pertaining to different process areas. The large number of projects allowed us to receive comprehensive feedback and to make wide-ranging observations. Furthermore, we were able to implement the approach in a wide variety of process areas. In line with our case selection goals, two of the case sets represented new development projects and one was a retrospective set. Table 5 gives an overview of the most important characteristics of the case sets. It shows the types of the projects belonging to each case set as well as the number of involved process models and activities. In the paragraphs below, we describe the details of each case set.

Case Set 1 e-Government. The e-Government case set was managed by the leading integrator organization for e-government projects in Turkey. The case set consists of a program comprising two projects. The program was initiated to develop two online systems for managing all processes related to the life cycle of companies (e-Company) and trademarks (e-Trademark) registered in North Cyprus. A team of three

external analysts worked on two projects in parallel, together with three internal analysts and two domain experts. In addition, 15 domain experts were occasionally involved in the workshops to provide domain-specific knowledge, but did not take part in the preparation and evaluation of the outputs. The internal analysts were experienced in different modeling notations, while the domain experts had only used natural language before and had no experience with process modeling notations.

Case Set 2 Public Services. The Public Services case set was managed by the Turkish Ministry of Development, the responsible institution for regional development agencies. In total, this case set consists of nine different projects. Among others, they cover the automation of public service processes provided by the development agencies, such as grant programs and investment support, but also internal processes such as human resource management. The team included three external and four internal analysts, together with four domain experts who took part in the preparation of the outputs, and 66 domain experts that were occasionally involved in analysis activities. The domain experts were not experienced in modeling notations and the internal analysts had only used flowcharts before.

Case Set 3 Campus System. The Campus System case set was managed by the Computer Center of the Middle East Technical University (METU), which is the top-ranked university in Turkey. The whole program consists of the automation of over 90 business processes, which concern the areas of research, education, campus services, and support. From this set, we selected two representative projects for this evaluation (Announcement and Research Program Management). The involved internal analysts were experienced in BPMN and other notations. In contrast the former two cases, this study took place shortly after the completion of the projects. With this retrospective case set, we specifically aimed to evaluate the completeness of the requirements generated via our approach in comparison to the requirements already defined with traditional approaches.

4.2. Implementation

We developed a prototype tool to facilitate the implementation of our approach in the case study. It is available as a plug-in for the integrated development environment *Eclipse* and based on the Eclipse Modeling Framework (EMF) and the Eclipse Graphical Modeling Framework (GMF).[1] The tool supports the development of process model diagrams in the EPC notation, the identification of automatable activities, the development of related requirements models in conformance with the exemplified FAD notation, and,

[1]The tool can be obtained from http:www.aysolmaz.com.

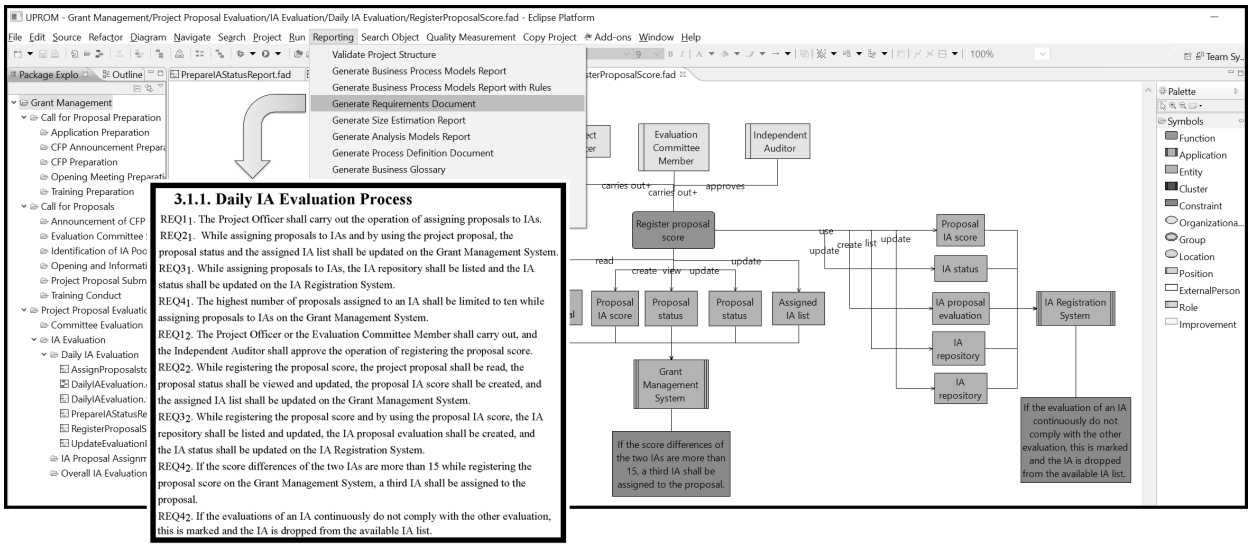


Figure 4: A screenshot of the prototype tool and the generated requirements document

lastly, the generation of textual requirements documents in conformance with the approach explained in Section 3. As all the cases were conducted in Turkey, we implemented the generation for the Turkish language. A snapshot of the (English) tool and the generated requirements document can be seen in Figure 4.

4.3. Conduct of the Case Study

Our case study consisted of three main steps: (1) the application of our semi-automated approach, (2) the analysis of the outputs, and (3) a set of feedback interviews.

The starting point of our case study was the *application* of our semi-automated approach by the project teams in the context of the cases. To make sure the teams could apply our approach in an effective and efficient way, we provided respective training to the teams before the start of the project. During the execution, we mainly acted as observers on how the teams applied the approach, but were also available for questions. Upon completion of the application and the generation of the requirements documents, we *analzyed* which changes were manually applied to the documents. After the completion of all project activities, we conducted a set of *interviews* with the internal and external analysts as well as with the domain experts who were involved in the projects. We chose interviews because they are the most prominent qualitative data collection method for obtaining in-depth insights [53]. Considering the number of team members involved in the case sets, interviews enabled us to develop a comprehensive understanding of the participant's experiences related to the use of our approach in a practical setting. The interviews followed

Table 6: Interviews for cases

Case set	Internal Analyst	External Analyst	Domain Expert
e-Government	2	3	2
Public Services	2	2	2
Campus System	3	1	

Table 7: Key figures of case study conduct

Case set	Project	#WS	EFF	#RM	#REQ
(1) e-Government	e-Company	10	76	82	363
	e-Trademark	6	41	36	177
	Total	***16***	***117***	***118***	***540***
(2) Public Services	Auditing	2	10	24	61
	Budget Management	28	154	339	822
	Archive Management	4	28	52	110
	Human Resource Management	12	46	159	336
	Investment	6	36	103	218
	Performance Management	14	67	18	36
	Program Management	3	23	72	154
	Project Support	41	148	457	1038
	Stakeholder Management	27	9	54	129
	Total	***119***	***539***	***1278***	***2904***
(3) Campus System	Announcement	3	6	18	65
	Research Program. Management	3	8	18	60
	Total	***6***	***14***	***36***	***125***

Legend: #WS = Number of workshops performed, EFF = Analysis effort in person-days, #RM = Number of requirements models, #REQ = Number of requirements sentences generated

a semi-structured style, taking around 45 minutes per interviewee. The interviews covered questions about the participants' background as well as the evaluation of our approach including the quality of the generated requirements documents. Table 6 shows the number of interviewees for each case set. We transcribed, coded, and analyzed the interviews to maintain a chain of evidence [52]. Table 7 summarizes the key figures of the case study performance. It shows the number of workshops performed (#WS), the total analysis effort spent (EFF), the number of requirements models developed (#RM), and the number of requirements generated based on these models (#REQ). In the following section, we discuss the findings of our case study.

4.4. Findings

In this section, we discuss how our semi-automated approach for requirements generation was assessed by the project teams of the three case sets. More specifically, we discuss how they evaluated four key characteristics that have been found to contribute to high-quality requirements [6, 39, 54, 55]: readability, completeness, consistency, and maintainability. We present our findings for each key characteristic separately for domain experts and analysts.

4.4.1. Readability

The readability (sometimes also referred to as *unambiguousness*) of a requirements document is one of its most important features [54].

Overall, all four *domain experts* found the documents informative and understandable. The domain experts who became familiar with process modeling through the training session and the workshops found the joint presentation of the process models and requirements sentences to further improve readability. For instance, one domain expert from case set 1 stated that "*Studying a model and the related statements together helped me to easily understand the requirements*". Other domain experts supported this with similar statements. Despite the generally positive feedback, some domain experts also mentioned aspects for improvement. For instance, one domain expert from case set 2 stated that the fixed structure of the sentences sometimes felt mechanical. At the same time, however, he also pointed out that such a generation facilitates a standardized and mature requirements structure.

All of the *analysts* mentioned that the generated documents were clear and understandable. Overall, they personally preferred to examine the models instead of the documents, but they found the generated documents to fit the purpose. An internal analyst from case set 2 stated: "*We needed to explain the system to various experts and the documents certainly helped us for this*". Some analysts also suggested specific changes to enhance readability. For instance, the team from case set 1 suggested to merge short responsibility and data need sentences into a single sentence. The same team also asked for removing the first part of the data need sentences ("*While* $<Action_{Gerund}>$"). We implemented the suggested changes by updating the generation algorithm respectively.

Altogether, we found that our approach generated well-readable requirements documents. In fact, all requested changes could be implemented by straightforward adaptations of the generation algorithm.

4.4.2. Completeness

The completeness of requirements is an important characteristic because it indicates the additional effort that has to be invested beyond the application of our approach [6].

The *domain experts* from case set 2 stated that the approach supported them to "*recognize whether the requirements are complete*". Overall, all domain experts agreed that the final set of requirements appeared complete. Besides that, they did not have further comments on completeness.

The *analysts* provided further comments on completeness. One internal analyst from case set 1, for instance, pointed out she "*would not be able to define such detailed requirements*" in another way. An analyst from case set 2 said that "*the approach was adequate to express what is required*". Emphasizing the support

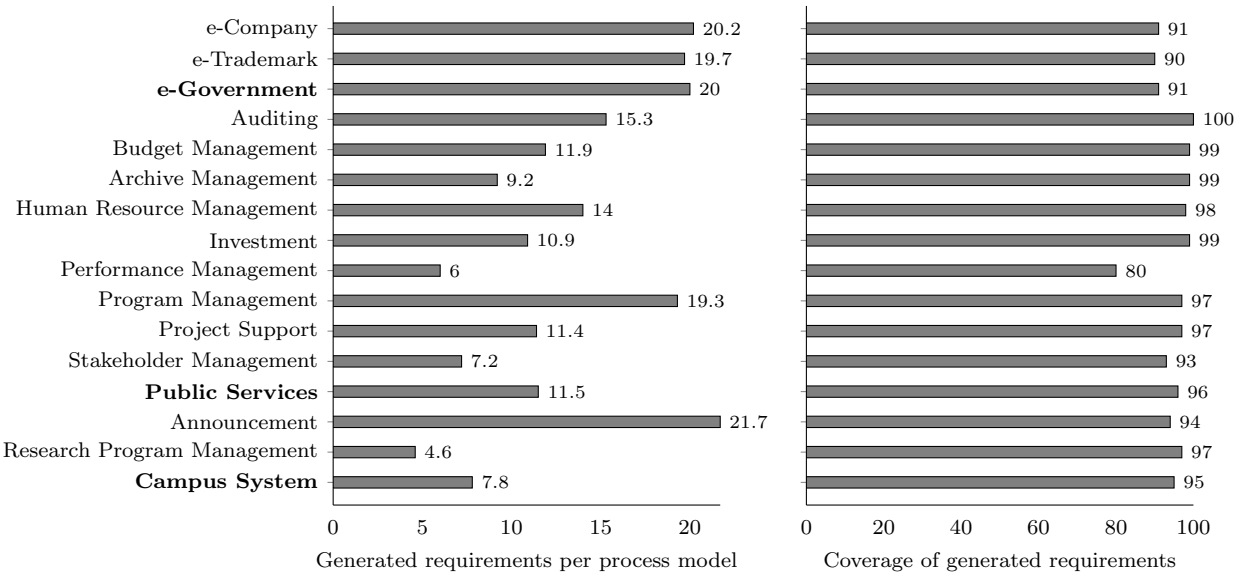


Figure 5: Generated requirements per process models and requirements coverage per case

provided by the approach, analysts also mentioned that our approach helped "*to collect the functional requirements with respect to the architectural components effectively*". Being asked for a comparison with traditional approaches, an analyst of case set 2 indicated that "*it would be harder to make a complete set like this if we wrote down the requirements textually in the first place*". He explained that "*we would miss a lot of aspects of the system if we didn't see the complete picture by means of the models*". Another analyst from case set 2 stated that the completeness was achieved as "*we were able to anticipate how the system should work as a whole and see the relations between different parts by means of the process-based requirements analysis*".

The overall completeness of the generated requirements for each project is illustrated in Figure 5. On the left-hand side, we can see the total number of requirements per process model and project. On the right-hand side, we see the coverage of the generated requirements for each project. For example, in the e-Company project of case set 1 about 20 requirement sentences were generated per process model and the generated requirements covered 91% of all requirements. The rest of the requirements were manually added by the analysts in case set 1 and 2. Among the manually added requirements, none related to the process-related aspects of the systems. Rather, they concerned general aspects which were not directly related to the processes and included the architecture of the system, interfaces with external systems, system-wide characteristics, security and quality requirements, and software development principles. Thus, they were not expected to be covered in the generated requirements set. Case set 3, the retrospective case, posed an

important role to evaluate the completeness. While in case sets 1 and 2 the requirements were developed from scratch in the context of the programs, case set 3 included an existing requirements document which was prepared in a different setting. We used the existing requirements as a benchmark and performed a delta analysis for the generated requirements. For this, we prepared a mapping between the existing requirement statements and the generated ones. The results showed that 95% coverage was achieved by the approach even with respect to the requirements already developed with traditional approaches. The unmatched requirements in the existing document related to quality aspects of the system. Moreover, six additional requirements were identified that were not included in the existing document. Thus, the findings of the retrospective case confirmed that a complete set of process-related requirements can be revealed by means of our approach.

4.4.3. Consistency

Consistency is another important characteristic of a requirements set and refers to the absence of contradictions within the set [55]. Our approach inherently ensures the consistency of the models and the natural language requirements by means of the automated generation approach.

From the *domain experts* we received very positive feedback with respect to the consistency. In fact, they explicitly stated that they did not observe any inconsistencies in the requirements.

The *analysts* were also very positive. They also had more specific comments on the achieved consistency. One internal analyst from case set 1 mentioned that "*especially if more than one person works on the analysis, this approach supports you to get the same quality of output from everybody*". Another analyst from case set 2 was initially critical about the usage of specific model elements for modeling the requirements, but later found that "*it was helpful for ensuring quality*". Here, it should be noted that the consistency of the generated requirements is dependent on the consistency of the models. In this respect, although the use of the approach does not ensure the consistency of the generated requirements, the model-based analysis helped the analysts to avoid such problems. All external analysts emphasized that "*updates would normally introduce consistency problems*", but that our approach helped to "*observe cross relations and to prevent resulting inconsistencies*". Internal analysts from case set 2 stated that they "*were able to define the requirements consistently although there were many different processes*" by means of "*the holistic view and the standardized language*".

4.4.4. Maintainability

Maintainability, sometimes also referred to as *modifiability*, is particularly important when it comes to changes [39]. All interviewees pointed out that they found the requirements easy to maintain. Among others, this was found to be caused by the improved traceability between process models and requirements.

One surprising finding was that even the *domain experts*, who typically do not develop models themselves, agreed on the improved maintenance. One domain expert from case set 1 stated that he "*could better understand the effects of a change*".

The *analysts* provided further discussions on how the maintainability was improved by our approach. One internal analyst of case set 1 stated that "*when a process was updated, it was also clear which requirements need to be changed*". While we also expected positive comments with respect to the maintainability resulting from the automated generation, we received quite unexpected feedback. All analysts stated that they did not find that the approach would save time to prepare the initial requirements document. One internal analyst from case set 1 stated that, creating the process models for the requirements generation is time-consuming: "*I would be faster with traditional methods, but I wouldn't be able to achieve the level of completeness*". Two external analysts of case set 1 and 2, as well as the internal analysts of case set 2 emphasized the potential time gain for updates and future development phases despite the extra time spent. Another external analyst from case set 2 stated that "*it may look like we spent more time, but in the long run, the time spent will be less*".

Overall, the interviews highlighted that domain experts were mainly interested in readability. Since domain experts often struggled with understanding the requirements, readability was their major concern. The analysts, by contrast, were also interested in the other three characteristics since they directly relate them to time savings and the automated support they expect from our approach. The analysts provided clear statements on how the approach enabled them to produce more complete, maintainable, and consistent requirements.

4.5. Comparison of Manually Created and Generated Requirements

The results of our case study showed that the generated requirements were positively perceived with respect to the four investigated key characteristics. An open question, however, is how exactly the manually created and generated requirements differ. To investigate this, we made use of the retrospective use case set *Campus System*. Our goal was to understand how the manually created and the generated texts compare with respect to text structure and how they convey the requirements content.

Table 8: Comparison of the text Structure of actual and generated requirements

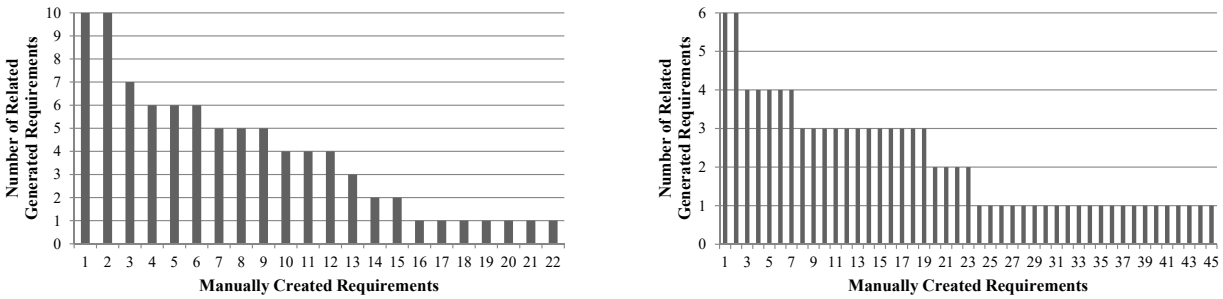
	Actual				Generated			
Project	**W/S**	**V/S**	**S/R**	**W/R**	**W/S**	**V/S**	**S/R**	**W/R**
Research Program Management	16.16	1.13	1.28	20.68	13.93	1.07	1	13.93
Announcement	14.19	1.43	1.60	22.77	13.48	1.38	1	13.48

To investigate the *text structure*, we computed a set of basic sentence complexity metrics [56]:

- Average number of words per sentence (W/S)
- Average number of verbs per sentence (V/S)
- Average number of sentences per requirement (S/R)
- Average number of words per requirement (W/R)

Table 8 summarizes the results of the comparison of the *text structure*. A general observation is that the generated sentences follow a similar structure like the manually created sentences, as indicated by similar values for the metrics W/S and V/S. This means that our approach generates sentences that are structurally comparable to those created by humans. However, we also observe some differences. Most notably, the manually created requirements contain a higher number of sentences and words per requirement (see S/R and W/R). This raises the question whether the manually created requirements are unnecessarily verbose or complex, which might explain the lower readability and comprehensibility perceived by users. A detailed analysis of the manually developed requirements indeed supports this conjecture. We identified many sentences in the manually created requirements that contained nonessential and repetitive descriptions. Among others, we found nonessential context information, redundant descriptions of functionality, and descriptions of of data attributes that were already defined in the data dictionary.

To understand how the manually created and the generated requirements convey their content, we mapped the manually created requirements to the corresponding generated requirements. Figure 6 visualizes



(a) Research Program Management (b) Announcement

Figure 6: Relationship between manually created and generated requirements

this mapping. It shows for each manually created requirement to how many generated requirements it relates. We observe that many of the manually created requirements relate to more than a single generated requirement. The average number of generated requirements per manually created requirement is 3.9 for the *Research Program Management* project and 2.2 for the *Announcement* project. Against the background of our findings from the *text structure* comparison, this is quite a surprising result. While the manually created requirements tend to be more verbose and, sometimes, even provide redundant information, the generated requirements document provides more details. We analyzed the extreme cases (i.e. where a manually created requirement relates to 10 generated requirements) and found that the manually created document lacked important details with respect to responsibilities and data needs.

In summary, we can say that this comparison highlighted the value of automated requirements generation. From a *structural point* of view, the generated requirements are very similar to the manually created requirements. The generated requirements, however, use less words and do not provide redundant information. From a *content* perspective, the comparison particularly illustrated the superiority of the generated requirements in terms of completeness.

4.6. Limitations

Despite the positive results, our evaluation has to be reflected from the perspective of some limitations. The first limitation relates to the *conducted interviews*. While the interviews allowed us to collect in-depth insights about the use of our approach in practice, interviews are also subjective by nature [53]. Among others, this means that the results of interviews could have been influenced by the bias of the interviewer. To avoid such a bias as far as possible, we designed and strictly followed an interview guideline. Moreover, an independent researcher reviewed the interview transcripts and confirmed the relevance of the answers with respect to the interview guideline. By following this procedure, we tried to minimize the limitations of interviews and obtain unbiased and reliable results. The second limitation relates to *generalizability of the overall case study* [52]. While we carefully collected a number of differing cases, we cannot claim that the results are representative or can be generalized to other organizations. However, since the feedback from the evaluation was consistently positive among the three cases, we are also confident that the presented approach can indeed provide considerable value for organizations.

5. Adaptation to Other Languages

From a conceptual perspective, the presented approach is not bound to a specific language. However, to use our approach for languages other than English, two main adaptations are required.

First, the templates must be translated and adapted to the target language. To illustrate the required steps, assume we would like to adapt the system to German and Turkish (i.e. two languages from different language families). What is more, reconsider the template "*The <Role> shall <Action> <Object>*" and its instantiation "*The Project Officer shall carry out the operation of registering the proposal score*". If we wish to adapt the system to German, we need to translate this template and adapt it to the German grammar. By replacing the word "*The*" with a new slot "<Article>" (in German the article depends on the gender of the referenced noun), by translating "*shall*" into "*soll*", and by switching the order of the action and the object slots, we obtain the template "*<Article> <Role> soll <Object> <Action>*". In a similar way, a respective template for Turkish can be obtained. Since Turkish does not use articles, the article "*the*" is omitted and the word order is adapted to the Turkish grammar. As a result, we obtain the template "*<Role> <Object>* $<Action_{Gerund}>$ işlemini $<Responsibility_{Verb}>$.".

Second, respective inflection mechanisms for the target language have to be implemented. In case of German this means that the correct article has to be determined based on the gender of the noun and that the verb must be conjugated. Both aspects can be achieved by using publicly available language processing tools such as SimpleNLG [57]. Based on this tool and respective German inputs for the slots, we are therefore able to generate a German version of the sentence: "*Der Projektleiter soll die Registrierung der Angebotsbewertung vornehmen*". For Turkish, only the gerund of the verb must be obtained. This can be achieved by looking at the last vowel of the verb and concatenating "*-ma*" in case of hard vowel sounds (e.g. a, u) and "*-me*" in case of soft vowel sounds (e.g. e, ü) to the end of the input verb. In this way, we are able to also generate a Turkish version of the sentence: "*Proje uzmanı teklif puanını kaydetme işlemini yürütecektir.*"

These examples illustrate that the adaptation of our technique is a one-time investment that is associated with reasonable effort. Because tools for inflecting words are available for many languages, only little technical knowledge about natural language generation will be required for the adaptation.

6. Implications

The approach we presented in this paper has several implications for research and practice.

From a research perspective, our work complements existing methods for requirements elicitation based on process models [4, 29, 30] by providing an automated way to obtain requirements documents. In contrast to existing approaches that consider automated support to elicit requirements, such as the ones proposed by Türetken et al [34] and Coşkunçay et al. [35], our approach was evaluated to generate requirements that

are well-readable, complete, and easy to maintain by means of the formulated requirements analysis and formalized natural language generation techniques. The consistency is ensured via automated generation. Our approach also informs methods for process model validation. In contrast to existing process model verbalization approaches [9], our approach also considers execution-related data and, thus, allows to obtain a more complete picture.

From a practical perspective, our approach helps to improve several characteristics that contribute to high-quality requirements, thus improving their usability. Other potential benefits for practitioners include the standardization of requirements engineering activities of analysts, enhanced testability, and improved scoping of the project. Hence, our approach can help practitioners in achieving considerable improvements in the software development process. While an extra effort must be spent in the initial analysis phase, the quality of the obtained requirements might save project teams from unnecessary repetitions in the SDLC. In the long run, our approach may, thus, also help to reduce costs. Taking these benefits into account, we believe our approach has the potential to influence the way requirements elicitation is conducted in practice. In fact, two organizations from our three cases, used the generated requirements document for finding a suitable software development subcontractor.

7. Conclusion

In this paper, we addressed the problem of inconsistencies between process models and natural language in the context of requirements specification. To cope with this problem, we introduced a semi-automated approach, which consists of two main phases. In the manual preparation phase, users identify the automatable activities in the input process model(s) and specify the associated responsibilities, data needs, system interactions, and execution constraints. The requirements model resulting from this analysis then serves as input for a generation algorithm, which automatically provides the user with a well-organized natural language requirements document.

We evaluated our approach by applying it in the context of a multiple case study with three organizations and a total of 13 projects. We found that our approach could be successfully applied to generate well-readable requirements that are complete, consistent, well maintainable, and, most importantly, of practical value. The interviewed analysts and domain experts pointed out that our approach positively contributed to the completeness, consistency, and maintainability of the requirements documents. Thus, the systematic analysis as well as the automated generation helped the studied project teams to deliver requirements documents of higher quality. This is emphasized by the fact that the generated requirements documents

were used for finding a suitable software development subcontractor in 11 of the 13 projects. Hence, we conclude that our approach successfully addresses the problem of inconsistency between process models and requirements documents, and provides real value to organizations.

In future work, we aim to extend our approach with the capability to automatically reflect changes of the generated requirements documents in the associated requirements and process models. In this way, the consistency between the artifacts can be also assured if changes are applied to requirements. Another aspect we wish to investigate is the specific impact of using the generated requirements. In this context, we plan to apply our method with and without the generated requirements documents. Besides that, we also plan to apply our approach in organizations that maintain English models. This will not only allow us to test our generation algorithm in another language, but also to evaluate the applicability in different cultures and settings. A final line of work we plan is to investigate the systematic transfer of the acquired requirements knowledge to the following software development phases. In this way, the benefits of the approach may also contribute to other phases of the SDLC.

Acknowledgement

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 660646.

References

References

[1] A. Gross, J. Doerr, EPC vs. UML Activity Diagram - Two Experiments Examining their Usefulness for Requirements Engineering, in: Requir. Eng. Conf. 2009. RE '09. 17th IEEE Int., 2009, pp. 47–56.

[2] C. Monsalve, A. Abran, A. April, Measuring Software Functional Size From Business Process Models, Int. J. Softw. Eng. Knowl. Eng. 21 (03) (2011) 311–338.

[3] J. Vara, M. Fortuna, J. Sánchez, C. Werner, M. Borges, A Requirements Engineering Approach for Data Modelling of Process-Aware Information Systems, in: W. Abramowicz (Ed.), Bus. Inf. Syst. SE - 12, Vol. 21 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2009, pp. 133–144.

[4] E. C. Cardoso, J. P. A. Almeida, G. Guizzardi, Requirements engineering based on business process models: A case study., in: EDOCW, 2009, pp. 320–327.

[5] K. Brennan, A guide to the Business Analysis Body of Knowledge (BABOK guide), version 2.0, 2nd Edition, IIBA International Institute of Business Analysis, 2009.

[6] IEEE, IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998, Tech. rep., Software Engineering Standards Committee of the IEEE Computer Society, Piscataway, N.J. (1998).

[7] Q. Ma, Y. Jiang, Process-oriented information system requirements engineering - a case study, J. Bus. Cases Appl. 10 (2014) 1–16.

[8] J. Li, R. Jeffery, K. H. Fung, L. Zhu, Q. Wang, H. Zhang, X. Xu, A business process-driven approach for requirements dependency analysis, in: Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), Vol. 7481 LNCS, 2012, pp. 200–215.

[9] H. Leopold, J. Mendling, A. Polyvyanyy, Supporting Process Model Validation through Natural Language Generation, IEEE Trans. Softw. Eng. 40 (8) (2014) 818–840.

[10] A. Coskuncay, An approach for generating natural language specifications by utilizing business process models, Msc thesis, Middle East Technical University (2010).

[11] T. Olsson, J. Grundy, Supporting traceability and inconsistency management between software artefacts, in: Proc. Int. Conf. on Software Engineering and Application, 2002.

[12] S. Winkler, J. Pilgrim, A survey of traceability in requirements engineering and model-driven development, Software and Systems Modeling (SoSyM) 9 (4) (2010) 529–565.

[13] H. van der Aa, H. Leopold, H. A. Reijers, Detecting Inconsistencies Between Process Models and Textual Descriptions, in: H. R. Motahari-Nezhad, J. Recker, M. Weidlich (Eds.), Bus. Process Manag. 13th Int. Conf. BPM 2015, Innsbruck, Austria, August 31 – Sept. 3, 2015, Proc., Springer International Publishing, Cham, 2015, pp. 90–105.

[14] R. S. Day, Alternative representations, The psychology of learning and motivation 22 (1988) 261–305.

[15] J. M. Polich, S. H. Schwartz, The effect of problem size on representation in deductive problem solving, Memory & Cognition 2 (4) (1974) 683–686.

[16] S. M. Schwartz, D. L. Fattaleh, Representation in deductive problem-solving: The matrix., Journal of Experimental Psychology 95 (2) (1972) 343.

[17] P. Wright, F. Reid, Written information: Some alternatives to prose for expressing the outcomes of complex contingencies., Journal of Applied Psychology 57 (2) (1973) 160.

[18] H. R. Ramsey, M. E. Atwood, J. R. Van Doren, Flowcharts versus program design languages: an experimental comparison, Communications of the ACM 26 (6) (1983) 445–449.

[19] T. G. Moher, D. Mak, B. Blumenthal, L. Levanthal, Comparing the comprehensibility of textual and graphical programs, in: Empirical Studies of Programmers: Fifth Workshop, Ablex, Norwood, NJ, 1993, pp. 137–161.

[20] D. A. Scanlan, Structured flowcharts outperform pseudocode: An experimental comparison, Software, IEEE 6 (5) (1989) 28–36.

[21] R. E. Mayer, Multimedia Learning, second edition, Cambridge University Press, Cambridge, UK, 2009.

[22] M. Weber, J. Weisbrod, Requirements engineering in automotive development-experiences and challenges, in: Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on, IEEE, 2002, pp. 331–340.

[23] B. Schätz, A. Fleischmann, E. Geisberger, M. Pister, et al., Model-based requirements engineering with autoraid., in: GI Jahrestagung (2), Citeseer, 2005, pp. 511–515.

[24] A. Davis, Just enough requirements management: where software development meets marketing, Addison-Wesley, 2013.

[25] J. Nicolás, A. Toval, On the generation of requirements specifications from software engineering models: A systematic literature review, Information and Software Technology 51 (9) (2009) 1291–1307.

[26] M. Dumas, W. V. der Aalst, A. ter Hofstede, Process-aware information systems : bridging people and software through process technology, John Wiley & Sons, New Jersey, 2005.

[27] M. Indulska, P. Green, J. Recker, M. Rosemann, Business Process Modeling: Perceived Benefits, in: A. Laender, S. Castano, U. Dayal, F. Casati, J. Oliveira (Eds.), Concept. Model. - ER 2009 SE - 34, Vol. 5829 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 458–471.

[28] H. C. Mayr, C. Kop, D. Esberger, Business Process Modeling and Requirements Modeling, in: Digit. Soc. 2007. ICDS '07. First Int. Conf., 2007, p. 8.

[29] O. Demirors, Ç. Gencel, A. Tarhan, Utilizing business process models for requirements elicitation, in: Euromicro Conf. 2003, 2003, pp. 1–4.

[30] C. Monsalve, A. April, A. Abran, Requirements Elicitation Using BPM Notations : Focusing on the Strategic Level Representation, in: 10th WSEAS Int. Conf. Appl. Comput. Appl. Comput. Sci., 2011, pp. 235–241.

[31] J. D. l. V. González, J. Díaz, Business process-driven requirements engineering: a goal-based approach, in: Proceedings of the 8th Workshop on Business Process Modeling, 2007, pp. 1–9.

[32] K. Cox, K. T. Phalp, S. J. Bleistein, J. M. Verner, Deriving requirements from process models via the problem frames approach, Inf. Softw. Technol. 47 (5) (2005) 319–337.

[33] S. Malik, I. S. Bajwa, Back to Origin: Transformation of Business Process Models to Business Rules, in: M. La Rosa, P. Soffer (Eds.), Bus. Process Manag. Work. BPM 2012 Int. Work. Tallinn, Est. Sept. 3, 2012. Revis. Pap., Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 611–622.

[34] O. Turetken, O. Su, O. Demirors, Automating software requirements generation from business process models, in: Proc. 1st Conf. Princ. Softw. Eng., Buenos Aires, Argentina, 2004, pp. 1–16.

[35] A. Coskuncay, B. Aysolmaz, O. Demirors, O. Bilen, I. Dogan, Bridging The Gap Between Business Process Modeling And Software Requirements Analysis: A Case Study, in: MCIS 2010 Proc., 2010, p. Paper 20.

[36] R. Davis, E. Brabander, ARIS Design Platform Getting Started with BPM, Springer London, 2007.

[37] T. Specht, J. Drawehn, M. Thränert, S. Kühne, Modeling Cooperative Business Processes and Transformation to a Service Oriented Architecture, in: Proc. Seventh IEEE Int. Conf. E-Commerce Technol., IEEE, Munich, Germany, 2005, pp. 249 – 256.

[38] C. M. Chiao, V. Kunzle, M. Reichert, Integrated modeling of process- and data-centric software systems with PHILharmonicFlows, in: Commun. Bus. Process Softw. Model. Qual. Understandability, Maintainab. (CPSM), 2013 IEEE 1st Int. Work., 2013, pp. 1–10.

[39] B. Berenbach, D. J. Paulish, J. Kazmeier, A. Rudorfer, Software & Systems Requirements Engineering: In Practice, Vol. 29, McGraw-Hill, 2009.

[40] L. Hunnebeck, ITIL Service Design, 2nd Edition, The Stationery Office, 2011.

[41] G. Hardy, Using IT governance and COBIT to deliver value with IT and respond to legal, regulatory and compliance challenges, Inf. Secur. Tech. Rep. 11 (1) (2006) 55–61.

[42] M. L. Smith, J. Erwin, Role and Responsibility Charting (RACI), Tech. rep., Project Management Forum (PMForum) (2005).

[43] COSMIC, The COSMIC Functional Size Measurement Method Version 4.0 Measurement Manual, Tech. Rep. April, The Common Software Measurement International Consortium (COSMIC) (2014).

[44] E. Insfrán, O. Pastor, R. Wieringa, Requirements Engineering-Based Conceptual Modelling, Requir. Eng. 7 (2) (2002) 61–72.

[45] S. Goedertier, J. Vanthienen, Declarative Process Modeling with Business Vocabulary and Business Rules, in: R. Meers-

man, Z. Tari, P. Herrero (Eds.), Move to Meaningful Internet Syst. 2007 OTM 2007 Work. SE - 83, Vol. 4805 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 603–612.

[46] E. Reiter, R. Dale, Building applied natural language generation systems, Natural Language Engineering 3 (1997) 57–87.

[47] E. Reiter, Nlg vs. templates, in: Proceedings of the 5th European Workshop on Natural Language Generation, 1995, pp. 95–106.

[48] K. V. Deemter, M. Theune, E. Krahmer, Real vs . template-based natural language generation: a false opposition ?, Comput. Linguist. 31 (2003) 15–24.

[49] H. Leopold, S. Smirnov, J. Mendling, On the refactoring of activity labels in business process models, Information Systems 37 (5) (2012) 443–459.

[50] D. Klein, C. D. Manning, Accurate Unlexicalized Parsing, 41st Meeting of the Association for Computational Linguistics (2003) 423–430.

[51] I. Benbasat, D. K. Goldstein, M. Mead, The Case Research Strategy in Studies of Information Systems, MIS Q. 11 (3) (1987) 369–386.

[52] R. K. Yin, Case Study Research: Design and Methods, 3rd Edition (Applied Social Research Methods, Vol. 5), SAGE Publications, Inc, 2002.

[53] J. Recker, Scientific research in information systems: A beginner's guide, Springer-Verlag Berlin Heidelberg, 2013.

[54] D. Firesmith, Specifying good requirements, J. Object Technol. 2 (4) (2003) 77–87.

[55] D. Zowghi, V. Gervasi, The three cs of requirements: consistency, completeness, and correctness, in: International Workshop on Requirements Engineering: Foundations for Software Quality, Essen, Germany: Essener Informatik Beitiage, 2002, pp. 155–164.

[56] X. Lu, Automatic analysis of syntactic complexity in second language writing, Int. J. Corpus Linguist. 15 (4) (2010) 474–496.

[57] M. Bollmann, Adapting simplenlg to german, in: Proceedings of the 13th European Workshop on Natural Language Generation, Association for Computational Linguistics, 2011, pp. 133–138.