On the Nature of Links between Requirements and Architectures: Case Studies on User Story Utilization in Agile Development

S. Molenaar

S. Brinkkemper

A. Menkveld

T. Smudde

R. Blessinga

F. Dalpiaz

Technical Report UU-CS-2019-008
August 2019

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

On the Nature of Links between Requirements and Architectures: Case Studies on User Story Utilization in Agile Development

Sabine Molenaar, Sjaak Brinkkemper, Abel Menkveld, Thijs Smudde, Remmelt Blessinga, Fabiano Dalpiaz

***Abstract*—Communication between requirements engineers and software architects is experienced as problematic. In this paper we present the Requirements Engineering for Software Architecture (RE4SA) model as a tool that supports the communication between these two roles. In the RE4SA model, requirements are expressed as epic stories and user stories, which are linked to modules and features, respectively, as their architectural counterparts. By applying the RE4SA model to a multi-case study, we investigate the nature of the relationships between the requirements and the architectural artifacts. Based on the gained experience, we put forward nine hypotheses for further research on the utilization of user stories in agile RE.**

I. Introduction

Communication flaws within a development team are considered one of the most important issues in requirements engineering (RE) and are sometimes identified as the main cause for project failure, according to the NaPiRE project [1]. In a 2014 study, Smith and colleagues found that 47% of unsuccessful projects failed due to poor requirements management [2]. Similarly, volatile requirements have been called one of the main issues in the software industry in recent history [3], [4]. The 'evil circle' principle [5] states that problems in the RE process do not remain isolated, but rather echo throughout a development process. Other major contributors to project failure include scope creep due to inadequate requirements, poor communications within the project team or among the stakeholders, and key internal stakeholders leaving the project [6]. Finally, proper architecture documentation can help prevent architectural drift and erosion, reduce costs and improve software quality [7].

Nuseibeh recognized that requirements specification and design cannot be separated due to their inter-dependencies [8]. The Twin Peaks model describes how requirements and architecture are defined concurrently, yet being separate specifications, with the former guiding the latter and the latter constraining the former. The Reciprocal Twin Peaks extends this work for agile development and explains why the synergy between requirements and artifacts matters. In short, a development process has to manage a continuous flow of requirements, as well as a continuously changing architecture [9]. Consistency between the two helps prevent misunderstandings in the development team. However, realizing this consistency should not burden the involved stakeholders with excessive work as the improvement in communication is meant to prevent incorrect implementations, rework and wasting time, money and potential other resources. Therefore, tools are needed to achieve consistency between the artifacts. Software architecture demands good requirements engineering, which can only be guaranteed if proper communication exists.

While Nuseibeh and Lucassen identified challenges and explained how RE and SA can support each other, they did not provide any specific approaches for tackling these challenges. As a remedy, we present explicit concepts and relationships that can be utilized to link requirements and architectures.

The remainder of this paper is structured as follows. In Section II, we present the RE4SA model, alongside its theoretical background, rationale and objectives. Subsequently, we discuss the feasibility and applicability in Section III by means of a multi-case study. This section also explains how the model can be applied, our main findings and the observed benefits. The empirical work serves as the basis for us to draw nine hypotheses that will guide future work (Section IV). Finally, Section V summarizes our contribution and discusses the main challenges we have identified.

II. The RE4SA Model

In an attempt to facilitate good communication within the development team, we propose the Requirements Engineering for Software Architecture (RE4SA) model, visualized in Fig. 1. RE4SA was assembled on the basis of tight collaboration with industrial partners in the software products domain, and it combines artifacts (like user stories and features) that we found often employed in their work practices.

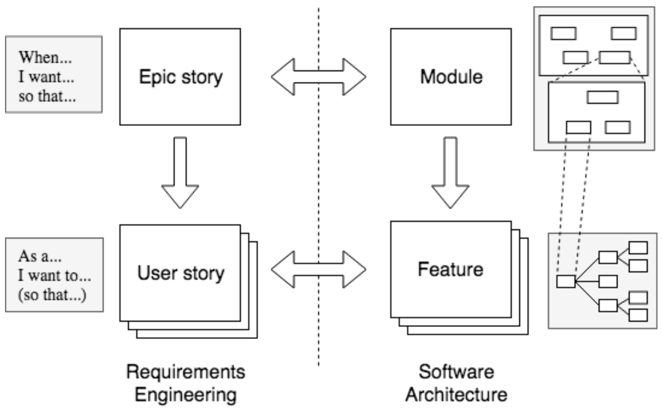


Fig. 1. The Requirements Engineering for Software Architecture model.

A. *Concept*

Similar to the Twin Peaks model, the RE4SA model links the RE process of a software product to its Software Architecture (SA). More specifically, it describes the links between Epic Stories (ESs) [10] and User Stories (USs) [11] in the requirements and modules and features in the functional architecture [12], respectively. Essentially, the problem space, which describes the requirements and their intended behavior, is related to the solution space that defines how intended behavior is implemented in a system and thus how requirements are satisfied [13]. ESs can be used to describe the modules in the architecture, while USs introduce more detail by describing the features of a software product. We define a functional architecture as a description of "*the system by its functional behavior and the interactions observable with the environment*" [12]. Examples of functional architectures are illustrated in the top two levels of Fig. 4 and Fig. 5.

B. *Illustration*

As an example, consider a navigation app. Some of its implemented features (white boxes) are visualized in Fig. 2. An ES, consisting of the three parts problematic situation, motivation and expected outcome, can be written to extend the functionality of the app: "*When I have to file for expenses to my employer, I want to have all my routes and local expenses registered, so that I can collect my expense data and minimize effort for filing.*" Using this ES it is possible to include the "Expense Filing" module in the software architecture. Going into the requirements process, the aforementioned ES can be refined by the following two USs. "*As a consultant, I want to file my travel expenses, so that I have complete expense and travel data with minimal effort*", using the verbs and nouns of the US this results in the "file travel expense" feature, which is part of the "Expense Filing" module. Subsequently, "*As a consultant, I want to collect my expenses of the past month, so that I can get an overview of my monthly costs*" can also be added. In addition, the existing functionality of the app needs to be extended in order to support these new features. In its current form, the app does not keep track of the user's previous routes, which is necessary in order to file the expenses. The following US can be added to remedy this: "*As a consultant, I want to file a route, so that I can calculate the costs of that route for my employer*". The three newly added features described by the previous ES and USs are illustrated as gray boxes in Fig. 2. Features are not only grouped based on their functionality, their naming can also be utilized to determine their position. For instance, the "file route" feature is part of the "determine route" composite feature within the "Route Planner" module. The keyword here is 'route'. On the other hand, the architectural elements related to the newly introduced ES all contain the word 'expense' or a variation thereof. Figure 2 illustrates the way ESs and USs can be transformed into functional architecture components and also serves as a simple means to discuss the position of the new functionality as well as the impact on the current implementation of the system.

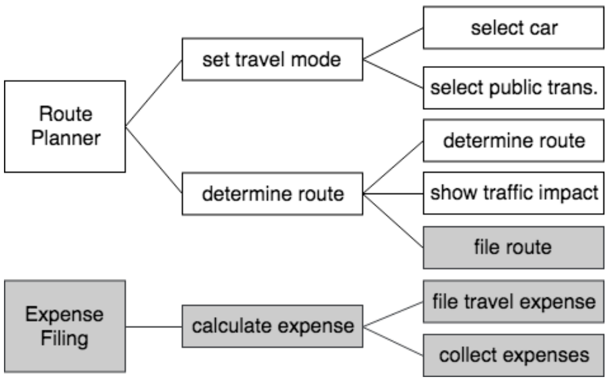


Fig. 2. Example of a feature diagram extension.

C. *Expected Benefits*

RE4SA is intended to improve communication between product managers and software architects or product owners through (1) simple communication means, (2) clear structural guidelines, and (3) consistent domain terminology. The objective of the RE4SA model, however, is not limited to improving communication. Gayer *et al.* argued for the need of dynamic architecture creation. This architecture allows for traceability in order to make software more maintainable, changeable and sustainable [14]. By establishing a relationship between ESs and modules and USs and features respectively, traceability is supported, with little documentation and effort required. The conflicts between architects and requirements engineers has been the subject of research before, such as in the case of the RADAR tool [15]. RADAR supports requirements and architecture decision analysis in an attempt to reduce struggle and miscommunication among stakeholders. As opposed to designing a new modeling language and analysis approach, we apply knowledge and techniques that already have a high adoption in the RE4SA model, in order to minimize the need for change and training in industry. USs, for instance, were found to often be one of the requirements documents used in agile methods [16]. The validity and applicability of the RE4SA model are discussed based on three case studies, presented in Section III.

III. Case Studies: First Experiences with RE4SA

To support the envisioned relevance and benefits argued in the previous sections, we present an industrial multi-case study that illustrates how RE4SA can be applied and that helps us develop the model and formulate hypotheses for future research. The three cases encompass different apps; while the selection is based on industrial availability, all of them target business consumers, and each studies a different use case of RE4SA: (i) modeling requirements and architectures prior to developing software, (ii) extending an existing product, and (iii) recovering an architecture. Case study findings on RE4SA are numbered and presented in bold.

A. Modeling for a Start-up

The first case is concerned with a software start-up in the context of intelligent greenhouses. To support modeling activities for the start-up's software, the RE4SA model was applied, which resulted in the formulation of 31 ESs and 96 USs. Based on the formulated RE artifacts, a functional architecture was developed that consisted of 31 modules. During this activity, the start-up's founder observed that **Finding-A.1:** ES formulation leads to module identification. Prior to development, the artifacts were discussed with the system's stakeholder to determine their value. Most importantly, it became apparent that none of the artifacts alone provided sufficient information to about the system to be developed. However, when taken together, the artifacts sufficiently provide a comprehensive overview, which provides evidence on the synergies between RE and SA, leading to the finding **F-A.2:** RE and SA artifacts shall be used in conjunction in a synergistic fashion.

Furthermore, this case has produced additional insight into how RE4SA should be applied to a development process. Firstly, it is important to determine the level of abstraction in terms of ESs and USs. Information to fully define the abstraction level is lacking prior to development, so instead it is important to distinguish between the ES and US level. Moreover, an ES should categorize at least two USs, otherwise the formulation of the ES is superfluous. This implies that if an ES contains only one US, the ES is formulated on an incorrect level of abstraction and should be reformulated to fit the US template instead. This may seem trivial, but is crucial to the structure of RE4SA given the levels of abstraction. Once this distinction is established, ESs can be written, followed by USs. **F-A.3:** the level of abstraction should be established prior to developing the RE artifacts.

The ESs were found to be especially useful in the modeling and subsequent naming of modules in the functional architecture. Names for modules could often be derived from the nouns and verbs included in the ESs, which does not only simplify the modeling process, but also facilitates linking RE artifacts to the functional architecture, **F-A.4:** module names can be derived from ESs. Likewise, the verbs and nouns used to formulate USs can be adopted to name features (as illustrated in Section II-B), **F-A.5:** feature names ought to be derived from USs. The formulation of ESs is often functional in nature, which lends itself to mapping them to a functional architecture, while still allowing the requirements engineers to work from a problem-oriented perspective. In addition to designing modules, ESs can support the specification of information flows between said modules too, as shown in Table I.

The (problematic) situation, motivation and expected outcome formulated in an ES can be utilized to determine the input flow, module name and output flow respectively. **F-A.6:** ESs provide naming suggestions for all elements in a functional architecture: modules, input flows, and output flows. An ES can be translated into functional architectural elements as illustrated in Fig. 3.

TABLE I
INFORMATION FLOWS AND MODULES FORMULATED BASED ON ESS.

Epic Story	**Module**	
	Input flow	**Output flow**
When using a neural network, I want to gather and format data continuously, so that the AI can interpret the data.	Data Manager	
	sensor output, weather report, growth model	climate data
When there is new prediction data available, I want to run it through a trained neural network, so that I can make yield predictions.	Harvest Predictor	
	climate data	prediction
When I have a yield prediction, I want to plan the right course of action, so that I can set the right climate conditions.	Action Planner	
	prediction	instructions: humidity, light, CO2
When receiving a humidity instruction, I want to determine a course of action, so that I can control humidity systems.	Humidity Action Management	
	humidity instruction	ventilation on timer, pump on timer

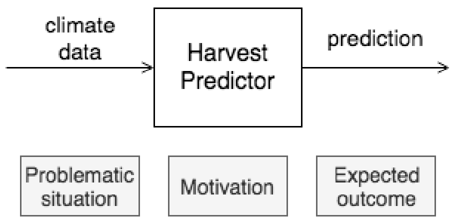


Fig. 3. ES to module translation in a functional architecture.

B. Extending a Software Product

The RE4SA model was also applied to a software company that determines the value of real estate and wanted to extend their software product with valuation analysis. The architectural artifacts were limited to tacit knowledge repositories, so it was unclear which existing modules and features were relevant for creating the software extension. Therefore, it was necessary to recover the functional architecture of the current system manually. The existing modules (28) have been modeled as feature diagrams, with 121 atomic features in total.

Extending the functional architecture with new modules resulted in several findings. First, the clarity of the visual representation was found to ease the communication between product manager and technical lead through the use of explicit architectural components. Second, the diagrams highlight which modules the extension depends on to implement new features. The development team confirmed that the functional architecture was helpful in discussions among stakeholders, leading to the finding **F-B.1:** the concepts in the RE4SA model are suitable for functional architecture recovery.

Then, the RE4SA model was applied to establish a functional architecture that satisfied the requirements of the software product extension. These requirements were elicited by the product manager, directly from customers of the software product. 23 USs were created and subsequently categorized in eight ESs. The functional architecture of the current system, illustrated in Fig. 4 positions the extension on three different levels of abstraction. These models identify parts

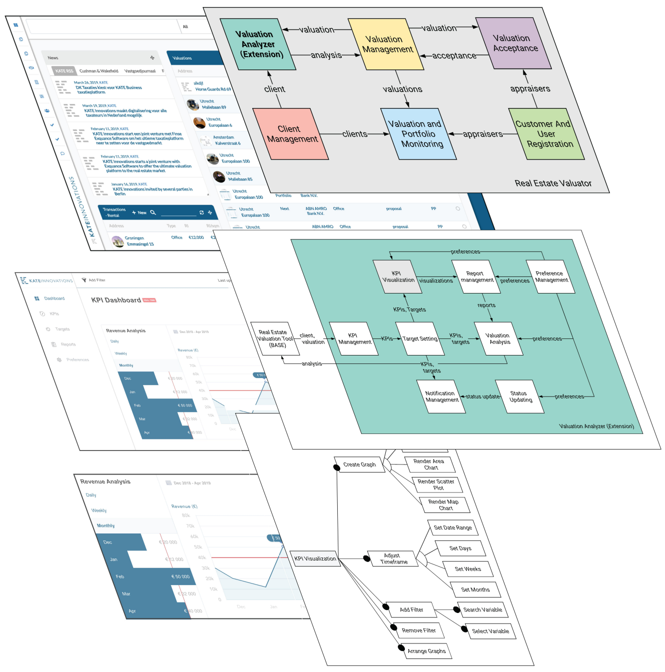


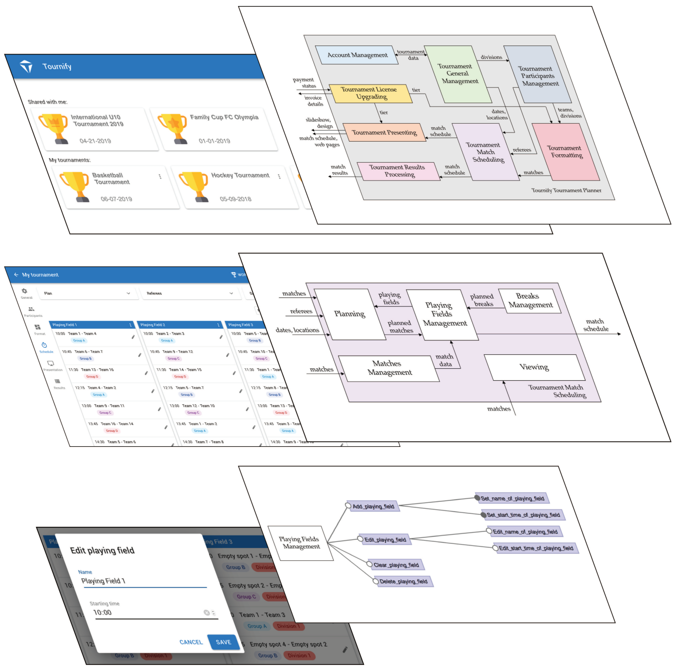
Fig. 4. Three-layered functional architecture of a real estate valuation tool.

the implemented system will be affected by the extension. Furthermore, the required interactions (modeled as information flows) with other modules are captured in the top level. The functional architecture enables a clear focus for sprint planning on developing well-defined components of the system and stakeholder validation of the architecture. **F-B.2:** the functional architecture allows sprint planning to focus on specific components. Moreover, thanks to the separation of concerns that RE4SA promotes, **F-B.3:** units of the software can be tested individually and thereby promotes re-usability. Furthermore, the functional architecture could be used to determine which parts of the system are (not) affected by the software product extension, **F-B.4:** the functional architecture has an appropriate level of abstraction that enables predicting the impact of new requirements on the existing system. Based on this evaluation of the RE4SA model in practice, there appears to be **F-B.5:** a 1-to-1 mapping between ESs and modules, and between USs and features.

C. Architecture Recovery of a Web Application

In a recent study, Tamburri and Kazman affirmed that "*both theory and practice suggests that maintaining good quality software architectures is non-trivial*" [17]. For web applications the problem may be even worse: proper documentation is rare, because well-known software engineering practices are seldom adopted by web developers and there is a high employee turnover rate [18]. To improve the understanding of these applications or systems, reverse engineering and system visualization techniques have been proposed [18].

The RE4SA model was applied to recover a functional architecture from the Graphical User Interface (GUI) of a web application. The application, a tournament planner with nearly 25,000 lines of code, was modeled in a feature diagram using the GUI as input for architecture recovery. Then, the website hierarchy and the principles of the RE4SA model were used to group the 199 atomic features into eight modules. Each module embodies a manageable and well-defined functionality that can be developed relatively independently from other modules [19]. Sub modules (21) were added for six modules to further differentiate between features and to facilitate the interpretation of the model by different stakeholders. The created interactive visualization of the model, shown in Fig. 5, was found by an interviewee at the company to help improve the communication between the stakeholders by allowing them to discuss specific components of the architecture, instead of a list with discussion points. Moreover, there is no need for all the stakeholders to understand the code. Therefore, we could conclude that **F-C.1:** the layered architecture recovered using the RE4SA model facilitates communication between stakeholders.



Tournify
Account Management
Tournament General Management
Tournament Participants Management
Tournament License Upgrading
Tournament Presenting
Tournament Match Scheduling
Tournament Formatting
Tournament Results Processing
Tournify Tournament Planner
Planning
Playing Fields Management
Breaks Management
Matches Management
Viewing
Tournament Match Scheduling
Edit playing field
Playing Field 1
CANCEL
SAVE
Playing Fields Management
Add_playing_field
Edit_playing_field
Clear_playing_field
Delete_playing_field
Set_name_of_playing_field
Set_start_time_of_playing_field
Edit_name_of_playing_field
Edit_start_time_of_playing_field

Fig. 5. Three-layered functional architecture of a web application, recovered based on the GUI.

During this case study, a crowdsourcing platform was used for the elicitation of new requirements in the form of USs. One user requested: "*As an organizer I want to set a unique start time of a playing field for each match day*". The current "set start time of playing field" feature does not allow this. If this feature would be implemented, it would require two new sub features ("set start time per day" and "set overall start time") with an alternative relationship to the feature above: exactly one of the sub features must be selected. **F-C.2:** the relationship between USs and features facilitates the positioning of new features in the functional architecture.

Mapping USs to features makes it easier to analyze which parts of the architecture are affected by evolving requirements. Furthermore, once the requirements have been mapped to the architecture, their location in the architecture and relation to the type of architectural component can support development estimations of USs. For instance, USs that are linked to atomic features were found to be relatively easy to implement by all stakeholders, while USs that require a new sub module are more complex and require more time to develop and implement. **F-C.3:** applying the RE4SA model facilitates impact forecasting in the context of changing requirements. Although no automation for creating and maintaining the traceability between the requirements, architecture and code was used, the case study shows how the RE4SA model can serve as a basis for communication and for the further analysis on the linguistic relationship between USs and features.

IV. Hypotheses

We have proposed a model that relates RE to SA in order to improve communication between stakeholders and support software development. The RE4SA model was proven to be promising based on a multi-case study, however, coincidence is still a factor and the relationships have not yet been sufficiently investigated. For instance, the strength of the relationships and their cardinalities are still unclear. Furthermore, the multi-case study that was presented previously was conducted on a small scale over a short period of time. To be able to accurately refine the links, large scale and long term research are required. In case of the latter, this could involve a study that examines the design and maintenance phase of a software development project, as opposed to one or the other. Moreover, we envision additional purposes for the model as well.

Based on the insight gained during the multi-case study, we have formulated nine hypotheses for future research in categories structure, requirements, architecture, and development process, presented in Table II. While the existence of relationships between the artifacts was confirmed, the exact nature of these relationships require further refinement as well as their cardinalities. The hypothesized cardinalities on the artifact structuring in the RE4SA model are shown in a meta-model in Fig. 6.

We hypothesize that ESs and modules should contain two or more USs and features respectively. If this rule is violated it is likely that an incorrect level of abstraction is used. Secondly, we expect that a US and a feature both belong to one ES or module. Based on the quality framework designed for USs, it is reasonable to assume that a US describes one feature, since a US of sufficient quality should express a requirement for exactly one feature [20]. Similarly, we expect an ES to describe only one module. On the other hand, we hypothesize that features and modules are described by one RE artifact, as we expect that the abstraction level may be inaccurate if

TABLE II
HYPOTHESES FOR FUTURE RESEARCH.

ID	**Hypothesis**	**Findings**
	Structure	
H.1	ESs and modules contain at least two USs or features.	A.3, C.1
H.2	There is a 1..1 relationship between ESs and modules and USs and features.	A.1, A.2, A.6, B.5
H.3	USs and features belong to exactly one ES or module respectively.	A.3, C.1
	Requirements	
H.4	The application of the RE4SA model effectively supports impact analysis of new requirements.	B.4, C.3
H.5	There exists a linguistic relationship between names of RE and SA artifacts.	A.2, A.4, A.5
	Architecture	
H.6	The RE4SA model supports architecture recovery activities.	B.1
H.7	The application of the RE4SA model effectively supports positioning of new features.	C.2
	Development process	
H.8	The RE4SA model can be utilized to guide and support testing activities.	B.3
H.9	The RE4SA model uses appropriate levels of abstraction so that it can be embedded in software product management activities, such as release planning.	B.2

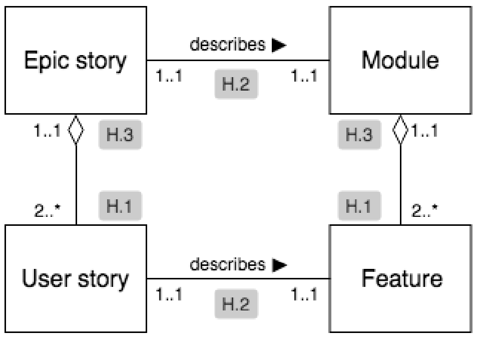


Fig. 6. Meta-model of the relationships between the RE4SA concepts, illustrating the hypotheses.

multiple RE artifacts are required to describe one module or feature. The hypothesized cardinalities (*H.1-H.3*) will be tested by conducting empirical research. Multiple case studies will analyze real-world artifacts in order to precisely define the links between the concepts.

A natural progression of this research is to analyze how change impact can be supported, especially in the context of impact forecasting and automated requirements traceability. *H.4* focuses on the maintenance and evolution phase of a software product and will therefore require a fully implemented system with existing development artifacts as a case study. Likewise, we aim to investigate how to generate (partial) artifacts to facilitate software development. We hypothesize (*H.5*) that both these objectives can be achieved by developing a complete picture of the artifact structures included in the model, as well as by conducting research on linguistic analyses of the artifacts and the potential discovery of linguistic patterns and links. These structures will be investigated using linguistic analysis techniques such as Part-of-Speech tagging and NLP.

Regarding architectures, functional architecture recovery was performed twice during the multi-case study presented earlier. However, this recovery was not the objective of the study and was also not performed in a structured manner. In order to properly test *H.6*, architecture recovery activities need to be performed (following a rigorous method) on a real-life case study and subsequently replicated using other cases. The RE4SA model can also be applied to design a method for software product design and development. This could be effective for keeping the software architecture up to date, and deciding where to position new features in the software. *H.7* will be tested by examining existing systems that need to be extended or updated through means of a case study. It is expected that during all of these studies, artifacts will need to be developed. Finally, case B and C have hinted at the RE4SA model's usefulness for release planning and guiding the testing process, stating that parts of the software can be tested somewhat independently and can provide separate functionality. *H.8* requires case studies to assess the applicability and feasibility of utilizing the RE4SA model for testing activities, as well as expert interviews or surveys to evaluate the usability and reliability. Finally, *H.9* will be tested based on additional literature research to determine how the model can be utilized and whether it uses the appropriate levels of abstraction, as well as case studies to validate these applications.

V. Conclusion

In this study on the links between requirements and architectures we propose a model with the objective to solve communication issues as well as supporting the software development process, best illustrated by the RE4SA model. A multi-case study was performed to verify the accuracy and applicability of the model in various contexts. The cases have also shown that the use of the model, and its underlying principles, supports multiple activities, such as: determining the level of abstraction for modeling the system, name derivations, identifying information flows, recovery of functional architectures, modeling extensions of an existing system, traceability between artifacts and impact forecasting. The most important results are the hypotheses for future research we formulated through the use of the multi-case study.

A few challenges related to the RE4SA model need to be addressed. Firstly, we need to study broader usage in large projects, since it was only applied in smaller cases up to this point. One of the main contributors to the expected hesitance of practitioners is the availability of a software architecture, more specifically, a functional architecture. On the other hand, the principle does rely on existing concepts that already have a wide adoption. As of yet we are not familiar with how the model can or should be applied in different development contexts. In similar fashion, there are no guidelines on how to apply the model. By this we mean that the starting point can vary and that the order of subsequent activities should not be fixed, since the development team should decide on the appropriate sequence of development activities. Finally, the software development process is not finished after eliciting requirements and designing the architecture. Future work should also focus on whether the RE4SA model can and should be extended in order to support more software product management activities, such as the design of technical architectures, feature programming, or release planning.

References

[1] D. Méndez Fernández *et al.*, "Naming the pain in requirements engineering," *Empirical software engineering*, vol. 22, no. 5, pp. 2298–2338, 2017.

[2] A. Smith, D. Bieg, and T. Cabrey, "PMI's pulse of the profession® in-depth report: Requirements management–a core competency for project and program success," *Project Management Institute, Newtown Square, PA*, 2014.

[3] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1288, 1988.

[4] D. Zowghi and N. Nurmuliani, "A study of the impact of requirements volatility on software project performance," in *Ninth Asia-Pacific Software Engineering Conference, 2002.* IEEE, 2002, pp. 3–11.

[5] T. Gilb and S. Finzi, *Principles of software engineering management.* Addison-wesley Reading, MA, 1988, vol. 11.

[6] D. L. Hughes, Y. K. Dwivedi, N. P. Rana, and A. C. Simintiras, "Information systems project failure–analysis of causal links using interpretive structural modelling," *Production Planning & Control*, vol. 27, no. 16, pp. 1313–1333, 2016.

[7] C. C. Venters, R. Capilla, S. Betz, B. Penzenstadler, T. Crick, S. Crouch, E. Y. Nakagawa, C. Becker, and C. Carrillo, "Software sustainability: Research and practice from a software architecture viewpoint," *Journal of Systems and Software*, vol. 138, pp. 174–188, 2018.

[8] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34, no. 3, pp. 115–119, 2001.

[9] G. Lucassen, F. Dalpiaz, J. M. Van Der Werf, and S. Brinkkemper, "Bridging the twin peaks: the case of the software industry," in *Proceedings of the Fifth International Workshop on Twin Peaks of Requirements and Architecture.* IEEE Press, 2015, pp. 24–28.

[10] G. Lucassen, M. van de Keuken, F. Dalpiaz, S. Brinkkemper, G. W. Sloof, and J. Schlingmann, "Jobs-to-be-done oriented requirements engineering: a method for defining job stories," in *International Working Conference on Requirements Engineering: Foundation for Software Quality.* Springer, 2018, pp. 227–243.

[11] M. Cohn, *User Stories Applied: for Agile Software Development.* Redwood City, CA, USA: Addison Wesley Professional, 2004.

[12] S. Brinkkemper and S. Pachidi, "Functional architecture modeling for the software product industry," *In: European Conference on Software Architecture*, pp. 198–213, 2010.

[13] S. Apel and C. Kästner, "An overview of feature-oriented software development." *Journal of Object Technology*, vol. 8, no. 5, pp. 49–84, 2009.

[14] S. Gayer, A. Herrmann, T. Keuler, M. Riebisch, and P. O. Antonino, "Lightweight traceability for the agile architect," *Computer*, vol. 49, no. 5, pp. 64–71, 2016.

[15] S. A. Busari and E. Letier, "Radar: A lightweight tool for requirements and architecture decision analysis," in *Proceedings of the 39th International Conference on Software Engineering.* IEEE Press, 2017, pp. 552–562.

[16] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in human behavior*, vol. 51, pp. 915–929, 2015.

[17] D. A. Tamburri and R. Kazman, "General methods for software architecture recovery: a potential approach and its evaluation," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1457–1489, 2018.

[18] A. E. Hassan and R. C. Holt, "Architecture recovery of web applications," in *Proceedings of the 24th International Conference on Software Engineering.* ACM, 2002, pp. 349–359.

[19] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[20] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, "Improving agile requirements: the quality user story framework and tool," *Requirements Engineering*, vol. 21, no. 3, pp. 383–403, 2016.