Specification of Requirements and Software Architecture for the Customisation of Enterprise Software

A multi-case study based on the RE4SA model

Tjerk Spijkman[§]*, Sjaak Brinkkemper*, Fabiano Dalpiaz*, Anne-Fleur Hemmer[§], Richard van de Bospoort[§]
* Dept. of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands
[§] Forza Consulting, Soest, The Netherlands
{tjerk.spijkman, anne-fleur.hemmer, richard.van.de.bospoort}@forzaconsulting.eu, {s.brinkkemper, f.dalpiaz}@uu.nl

Abstract— **Many failed software projects can be traced to bad requirements management. Additionally, there is a big gap between state of the art and practice in software architecture. For enterprise software customisation, not only do these issues apply, but additional challenges exist too. Instead of one standard software product, vendors often have to deal with customised versions with additional maintenance challenges. In this research, we apply the Requirements Engineering for Software Architecture (RE4SA) model via a multi-case study to show how the requirements engineering and software architecture disciplines can be linked, and in doing so provide improvements to both areas. Our multi-case study regards enterprise software customisation and shows improvements in requirements management and higher alignment between the software architecture and requirements.**

Index Terms— **Requirements Engineering, Software Architecture, Case study, Enterprise software, Software products, Customisation.**

I. Introduction

Requirements engineering (RE) is one of the key processes in the creation and customisation of software products, as it addresses the critical problem of designing the right software for the customer [1]. Poor requirement management can complicate software development projects, and lead to project failure. A survey conducted by PMI in 2014 shows that almost half project failures can be linked to poor requirements management [2]. By placing appropriate focus on requirements engineering, it is possible to prevent project failure, meet deadlines, effectively plan releases, facilitate communication, and ensure that a solution meets the stakeholder's needs [1-3].

The second concept that we study in this research is software architecture (SA). Bass, Clements and Kazman define SA as the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both [4]. Bass *et al.* also state that software architecture constitutes a common language for all stakeholders and captures design decisions in the early stages of a software product. As software products get bigger, good architecture design is required to ensure a loose coupling within the software, and to facilitate more effective collaboration [5]. Good architecture documentation is often scarce in practice, and there is a gap between state of the art and state of the practice in software architecture [6]. Therefore, there are clear opportunities to improve the use of software architecture in practice.

Nuseibeh's Twin Peaks model is one of the first attempts to link requirements to software architecture in order to improve both disciplines and to more effectively suit agile development [7]. Applying the model results in a stepwise, concurrent refinement of the requirements and the architecture, which adds details to both. Lucassen *et al.* extend this approach and introduce the Reciprocal Twin Peaks (RTP) model and discuss how to achieve alignment between RE and SA in practice [8]. They also analyse how this approach is different for product software compared to tailor-made software, due to the necessity to accommodate requirements that originate from many customers.

These requirements can lead to multiple different implementations of a standard enterprise software product. The majority of enterprise resource planning (ERP) project costs are devoted to software setup, installation and customisation [9]. While one benefit of enterprise software is the provision of a standard solution, they often need to be customised to support specific business processes or meet the company's needs [10]. In 2013, Panorama consulting [11] reported that 90% of enterprise software had at least minor customisations. There are different categories of customisation. Software can be modified, by changing existing functionality; extended, by adding functionality to an existing module; or additional modules can be added to the software product. These customisations can cause issues when a new version of an extension is deployed, or when the enterprise software is updated. [12, 13] Fig. 1 illustrates these customisation types on a sample architecture.

One recent, pragmatic approach to help align RE and SA is the RE4SA model [14], which consists of links between specific artefacts in requirements engineering and software architecture. The model differs from feature driven design (FDD) [15] in that it focuses on the creation of functional architecture in parallel with the design as opposed to the more technical UML models used in FDD, and by directly linking concepts of the requirements to a functional architecture. RE4SA also provides a less abstract view compared to architecture centric Extreme programming [16]. Practitioners often view architecture-centric methods as excessive work [16], and RE4SA comes to help by proposing specific links between concrete artefacts and by suggesting the derivation of an architecture from the already existing requirements place.

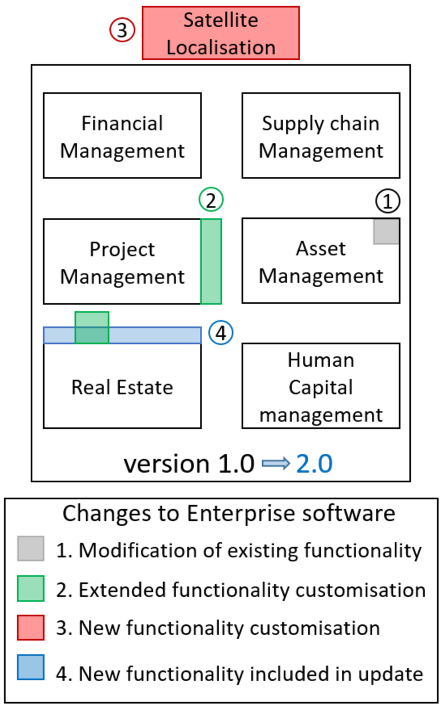
③
Satellite Localisation
Financial Management
Supply chain Management
②
①
Project Management
Asset Management
④
Real Estate
Human Capital management
version 1.0 ⟹ 2.0
Changes to Enterprise software
1. Modification of existing functionality
2. Extended functionality customisation
3. New functionality customisation
4. New functionality included in update

Fig. 1. Customisation types for a software product

The RE4SA model is an active field of research at Utrecht University. Although initial findings on the use of this model are promising, there is a lack of evidence of its effectiveness in practice. In this research, we apply the RE4SA model to a multi-case study on four cases of a specific product software solution. We aim to validate the use of the model in a real-world setting, and to further build the theory by observing how the model can provide improvements to practitioners. To such extent, we define the following research questions:

RQ: How can RE4SA be applied to improve communication and documentation for customisations of enterprise application software?

As this research aims to build and test RE4SA and its generality in different situation, we purposefully choose four different use cases instead of focusing on near-identical cases. We focus specifically on the application of RE4SA to enterprise software. These products often lack architectural design and documentation, receive requirements from different customers, and have customer-specific implementations [8, 14].

The literature on the customisation of enterprise software is limited, and we attempt to fill this gap, by testing the RE4SA model and reporting on its use in practise. We share the results of our case studies, and in doing so propose an improvement to current processes, provide an effective way for requirements management and leverage the opportunities for applying functional software architecture in practise.

The rest of this paper is structured as follows. Section II details the RE4SA model. Section III introduces the context of the case study. Section IV reports on the main findings of the case studies. Section V details the findings from expert evaluations of the case studies and list the lessons learned through the case studies. Finally, Section VI discusses the findings, and puts forward research directions.

II. The RE4SA Model

The Requirements Engineering for Software Architecture (RE4SA) model aims to align the RE and SA disciplines. Unlike the Reciprocal Twin Peaks model [8], it does not link the responsibilities of actors within the RE and SA disciplines, but rather proposes links between specific artefacts. This allows to base the software architecture directly on the requirements gathered for the product and ensure that the requirements are met by the software. Additionally, RE4SA suggests specific trace links between RE and SA that can facilitate communication between team members, and customers.

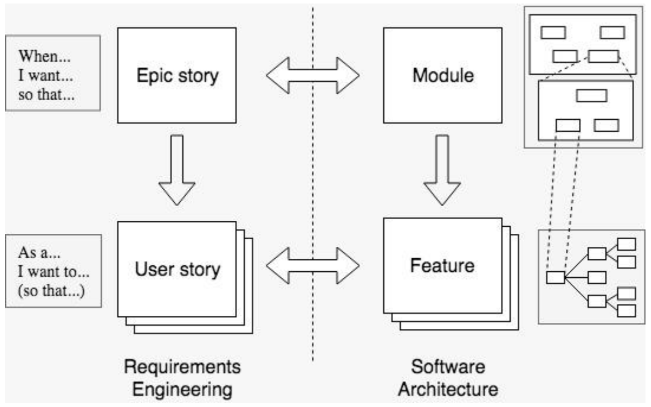

Fig. 2. Illustration of the Requirements Engineering for Software Architecture (RE4SA) model [10]

A. *Explanation of the Model*

The model is especially promising because it utilizes user stories, a requirements notation that is widely used in practice [17, 18]. Because of the high adoption of user stories in practice, this model allows for the use of available documentation to keep an up to date software architecture. In this model we link epic stories (ES) [19] to modules [20] in the software, and user stories (US) [21] to features [20]. ES are based on the job stories introduced by Klement [22], which were renamed at Utrecht University to Epic Story because the term Epic is already existent in Scrum.

Epic stories can be used to detail the requirements that should be solved in functional modules of a software product, while US can be used to detail the requirements for more specific features within a module.

Among the expected benefits of the model, we foresee it facilitates alignment between RE and SA, improves communication and collaboration, helps with release planning, prevents architectural drift, provides traceability from requirements to solutions, and delivers concise yet detailed documentation.

The links between RE and SA artefacts can provide a reciprocal benefit between the two disciplines. In the context of enterprise software, we can visualize where customer specific changes are located in the software, saving time in a later project (for example an update) at the same client. An overview of the dependencies for a customisation can provide valuable information in the risk assessment of a customisation. In this context, the model can also be applied to plan future releases.

B. *Example: the RE4SA Model Applied*

To illustrate the RE4SA model, we present an example that shows how it helps link the artefacts. This example is based on the case study that we conducted, and highlights a small section of the software product analysed for the case studies.

Consider the following epic story for an invoice automation application: *"When there is an issue with an invoice, I want a way to contact another user, so that the issue can be resolved by the relevant user."* This requirement was addressed via a "QA Form" module in the application. Each epic story contains multiple user stories, one such US for this example could be: *"As an approver, I want to set a subject for my question, so that the person who I ask the question can quickly see what it is about."*, which can be addressed through a "set subject" feature in the QA module. The RE4SA model with the artefacts for this example can be seen in Fig. 3; note that this figure shows only a fragment of the architecture and the requirements to illustrate the example.

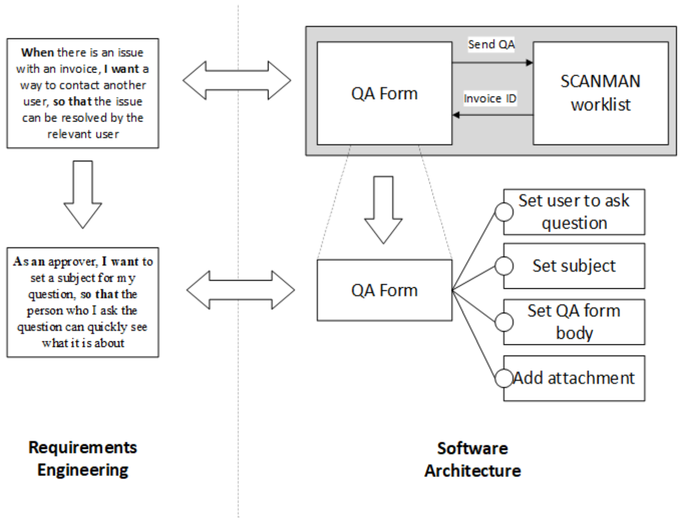
When there is an issue with an invoice, I want a way to contact another user, so that the issue can be resolved by the relevant user
As an approver, I want to set a subject for my question, so that the person who I ask the question can quickly see what it is about
QA Form
Send QA
Invoice ID
SCANMAN worklist
QA Form
Set user to ask question
Set subject
Set QA form body
Add attachment
Requirements Engineering
Software Architecture

Fig. 3. RE4SA applied to the SCANMAN QA example

III. Case study context

In this research, a case study was performed in which we analysed four different cases within a single company. The case study protocol was based on the guidelines as described by Runeson and Höst [23]. The results from applying the RE4SA model in the cases was evaluated through 11 expert interviews. In this section, we discuss the goals of the four cases and we introduce the case company and the software product that is central for each of the cases.

The goal of the case studies was to identify how the RE4SA model can be applied, and to analyse the benefits of applying the model in the context of enterprise software customisation, and management (see RQ1-RQ3). Each of the cases describe the application of the model to a project for the SCANMAN software product (see Table I). For each of the cases, we created all the RE4SA artefacts based on available documentation and evaluated this through expert interviews. By performing case study research, we test the applicability of the model to a range of cases and use the outcomes as an input for expert evaluations of the model. Based on our research question we aimed to test if the RE4SA model can be applied:

- to a broad range of projects for a software product;
- to effectively and efficiently document requirements and software architecture;
- to provide an overview of customer specific functionalities in an environment; and
- to facilitate communication between functional and technical experts.

Table I. Overview of the four cases

ID	Description	# Interviews	Interviewees
1	Customisation at the feature level	2	Developer, Deployment expert
2	Customisation at module & feature level	3	Functional consultant (x2), developer
3	Version update	3	Developer (x2), management
4	Recreation of the ES addon	3	Project manager, functional consultant, developer

A. *The Company*

Forza IT group is a company that is mainly focused on supporting their customers in using Oracle enterprise software. The company has offices in Soest, The Netherlands and in Sofia, Bulgaria. They provide consultancy services, perform ERP implementations, and develop extensions to the enterprise software. This means they are involved in both the RE and SA fields, as they set up the ERP environment to match the customers' requirements. Occasionally, they make changes to the software so that it fits the customers' business processes. Besides the standard Oracle software, they also develop their own add-on solutions to the software that extend the functionality of the ERP system.

B. *The Software: SCANMAN*

One of Forza's most successful add-ons is SCANMAN. SCANMAN is an invoice automation application that is fully integrated within the JD Edwards (JDE) ERP system. SCANMAN uses Optical Character Recognition (OCR) to scan incoming invoices, and to automatically enter these values into JDE. These values can then be validated, and the add-on also adds an acceptance flow to JDE, in which users can accept or reject an invoice.

For SCANMAN, Forza obtains a high number of requests for functionality/features from customers. In most cases, these new functionalities/features become part of the standard version of SCANMAN, but some of the additions are customer specific. These customer-specific functionalities cause additional challenges. These customisations need to be retrofitted in upgrades, can cause conflicts with new version of the ERP, and cause other maintenance issues. For example, during an upgrade, it first needs to be identified what customer specific changes were made, and then the decision to recreate the customisation or leave it out needs to be made based on the circumstances. In these situations, it would be very beneficial to have an overview of the software architecture so that identifying customisations can be made easier.

IV. Case Study Findings

We describe the context and the main findings for each of the four cases. As a prerequisite for the cases, we first had to recover the software architecture of the current version of SCANMAN, and the functionalities of JDE that it relies on. This recovery was done manually by the researchers, through an analysis of the tool, its documentation and the user guides. This resulted in a functional architecture diagram (FAD) and a feature diagram. The FAD shows the modules of the application, and how these interact [20]. The feature diagram shows the user-visible aspects of the software system [24] and was recovered by modelling all aspects of the GUI. Since the feature diagram was recovered from the GUI, the features are likely at a more detailed level than feature diagrams created to define a software design. By focusing on the GUI, we can recover nearly all functional features as these are often represented in the GUI. The recovered architecture contained 62 (sub-) modules, and the feature model contained 1,340 features. The case studies introduced additional elements to all of these artefacts The epic stories and user stories mentioned in the case studies were written manually and validated with experts at the case company.

A. *CS 1: Tracing Customer Requirements to the Architecture*

In this retrospective case study, we applied the RE4SA model to a requirement set sent to the case company by a recently acquired SCANMAN customer. The set of 46 requirements were rewritten to 32 functional user stories, and 7 quality requirements. Only two of the requirements detailed functionality that was not in the current software. The quality requirements were tracked in JIRA and left out of scope for this case study. None of the requirements were on an ES or module level, so only the lower part of the RE4SA model was used.

For this case study, we traced each of the requirements to the corresponding features and marked these in green. This colour coding allowed for quick identification of the relevant features, which was necessary due to the high number of total features. The new features were added to the feature diagram and colour coded in red to signify that they represent customer-specific additions. The RE4SA model proved useful for deciding the appropriate location for the new features, for we could use the user stories to locate relevant features as seen in Fig. 4.

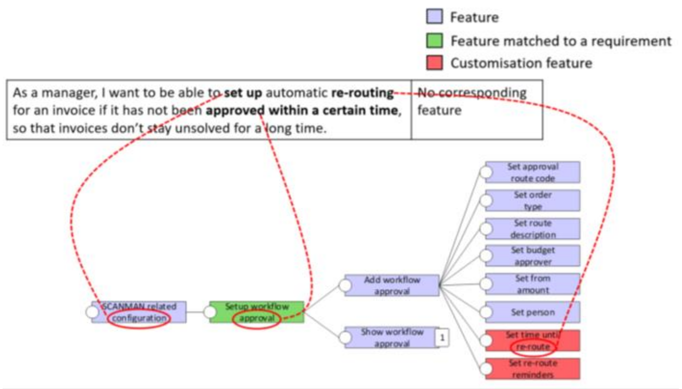

Fig. 4. New feature placement based on a user story

In this case, we found that once the "standard" feature diagram is made, this can easily be copied and extended to capture the customer specific environment. The traceability between requirements and architecture can be utilitzed to identify an appropriate placement for new features. Since the feature diagram is rather complex and has a high number of features, this complicates the use of the model for communication. However, by collapsing irrelevant modules, or using only a small section of the model, like in Fig. 4, can mitigate this issue and makes the model useful for communication purposes. Furthermore, colour coding specific elements, like customer specific features, makes it possible to easily spot important features in the model.

B. *CS 2: Customer specific module and features*

The second retrospective case was chosen because it had a customisation that was on the ES / module level. For this case, all JIRA issues concerning the specific customer were analysed and transformed to user stories. This resulted in 1 new ES, and 14 US; these numbers are lower than those of case 1 because only new functionalities were included in the JIRA tickets, and not existing ones like those matched in the first case.

The ES that was written for this case was *"When I receive an invoice that does not match an existing purchase order, I want to receive a report when it exceeds the tolerance set in the ERP, so that the invoice can be rejected."* This resulted in the user visible sub-module VMA comparison report. This shows one limitation of the GUI-based feature diagram, which depicts only the features visible to the user, and not the processes "in the background" that are required to generate the report. The ES for this case contained 7 user stories, and the VMA report consists of 23 features. This difference in cardinality can have a number of possible reasons: (i) the included information field already existed within the ERP so all relevant fields might be selected which results in a higher number of fields than required, due to limited context knowledge for the case study some customer requirements could have been missed in the US, or (ii) some features might be considered 'common knowledge' and do not need a user story to be included in the solution.

In this case, we extended the colour coding to the FAD, thereby allowing for a quick overview of which modules have been altered for the customer environment. Note that this information can be used to facilitate the information seeking mantra [26]. The colour coded FAD for this case, with drill-down navigation to the feature level can be seen in Fig. 5, where red was used to signify customised modules, cyan for JDE modules that the application relies on, and orange as a warning because that JDE module is required specifically for a modification.

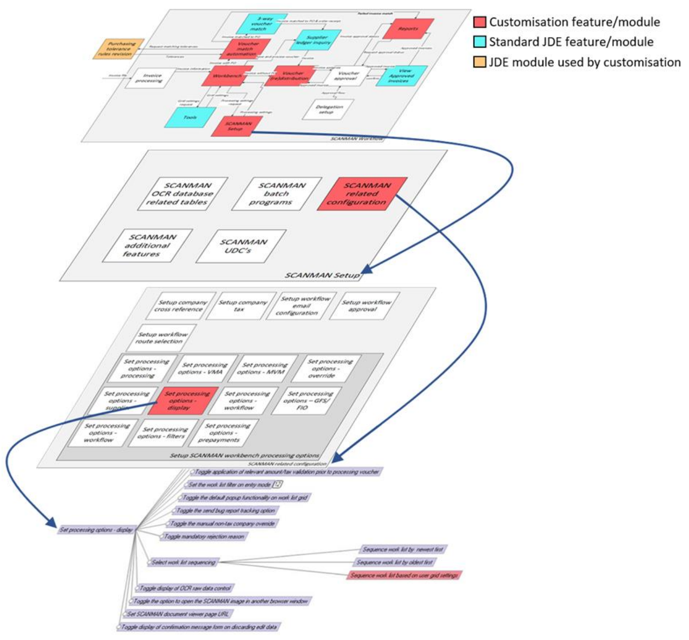
Customisation feature/module
Standard JDE feature/module
JDE module used by customisation
Purchasing tolerance rules revision
Invoice processing
Workbench
Voucher match automation
Voucher (re)distribution
Voucher approval
Supplier ledger inquiry
Reports
View approved invoices
Tools
SCANMAN Setup
Delegation setup
SCANMAN OCR database related tables
SCANMAN batch programs
SCANMAN related configuration
SCANMAN additional features
SCANMAN UDC's
SCANMAN Setup
Setup company cross reference
Setup company tax
Setup workflow email configuration
Setup workflow approval
Setup workflow route selection
Set processing options - processing
Set processing options - VMA
Set processing options - MVM
Set processing options - override
Set processing options - display
Set processing options - workflow
Set processing options - GFS/FIO
Set processing options - filters
Set processing options - prepayments
Setup SCANMAN workbench processing options
Set processing options - display
Select work list sequencing
Sequence work list by newest first
Sequence work list by oldest first
Sequence work list based on user grid settings

Fig. 5. Colour coded software architecture with drill-down navigation

The colour coding allows for zooming from the FAD for the high-level modules to sub module(s) to features in the feature diagram. This drill-down navigation is currently only possible by manually opening views based on the naming in other views. Ideally, this process would be facilitated through a tool which links all the models together, allowing for navigation by zoom on click.

C. *CS 3: Version Update*

We applied the RE4SA model side by side with the current method in an ongoing project. The version update collects requirements from many different customers that are to be included in the standard version. Some of these additions are already developed in specific customer environments and need to be moved to the standard software product.

For the update case study, we documented a total of 3 ES (one of these is the QA functionality from Fig. 3) containing 16, 2 and 6 US, respectively. Additionally, 18 US that extend already existing modules were identified, which did not belong to an epic story. For these user stories we had a challenge to solve, as they could not be linked to a relevant ES since the ES for existing modules were never created. To mitigate this issue, we decided to apply the RE4SA model, and considered the link between ES and modules. This meant that instead of having to recover ES for existing modules, we could just use the name of the relevant modules to categorize these user stories.

While we had only 3 ES and 40 US, the feature diagram was updated with 104 new features and the FAD was updated with one new module and three submodules. The difference in ES and module is because two modules were added for the QA epic, a QA overview & specific QA module, this indicates that we either forgot to formulate an ES, or it was added due to goal focused nature of the requirements. Multiple explanations exist to explain the US-to-feature cardinality ratio of 1:2.6: (i) like in case 2 some features were added based on ‘common sense’ (e.g. fields in the reports); (ii) developers might have added unrequested features; and (iii) some user stories can be solved by a collection of features. An example of this can be seen in Fig. 6; this is not inherently a bad thing, as it ties into the US principle of focusing on goals [21]. This principle implies that we don’t care how we get to our goal, as long as we can get there. Fig. 6 also shows an example of ‘common sense’ features in the sub-features of the “show QA history” feature.

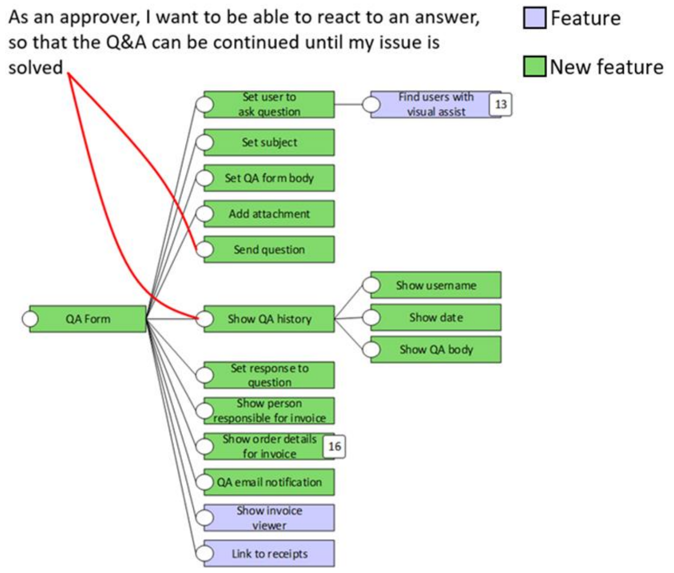

Fig. 6. A single user story which is addressed by two features

As the software architecture evolves in new versions and customisations, this also allows for adaptations to the architecture models. While adding the new features to the feature diagram, it was noticed that some existing functionalities were not yet included in the reconstructed architecture. For example, in this update new features were added to the invoice processing module, however in the process of adding these features to the feature diagram it appeared that the invoice processing module had not been included in the feature diagram yet. Because these new features extended a module that was not mapped, this error in the feature diagram could be detected and fixed. Thus, the application of the RE4SA model in multiple projects also allows for refinement of the architecture model and increases the accuracy of the models over time.

The case study report was used as a basis for the update notes of the version update. The software architecture provided a quick overview of all added functionalities, which was a lot less effort to analyse than all the logged JIRA issues. By using cut-outs of the relevant parts of the feature diagram (like Fig.

6), it becomes possible to detect the new functions within the software. Combined with a short textual explanation, the digram can provide enough information for the update notes.

D. *CS 4: Recreation of SCANMAN for NetSuite*

The fourth case study is somewhat different than the others, as it is not focused on the JDE version of SCANMAN, but on recreating the functionality of SCANMAN in NetSuite, another Oracle enterprise application. For this case study, we used the software architecture and documentation of the JDE version to discover the core functionalities and to decide what needed to be included for the first version of SCANMAN NetSuite. This case can still be seen as a customisation, as it extends the functionality of NetSuite.

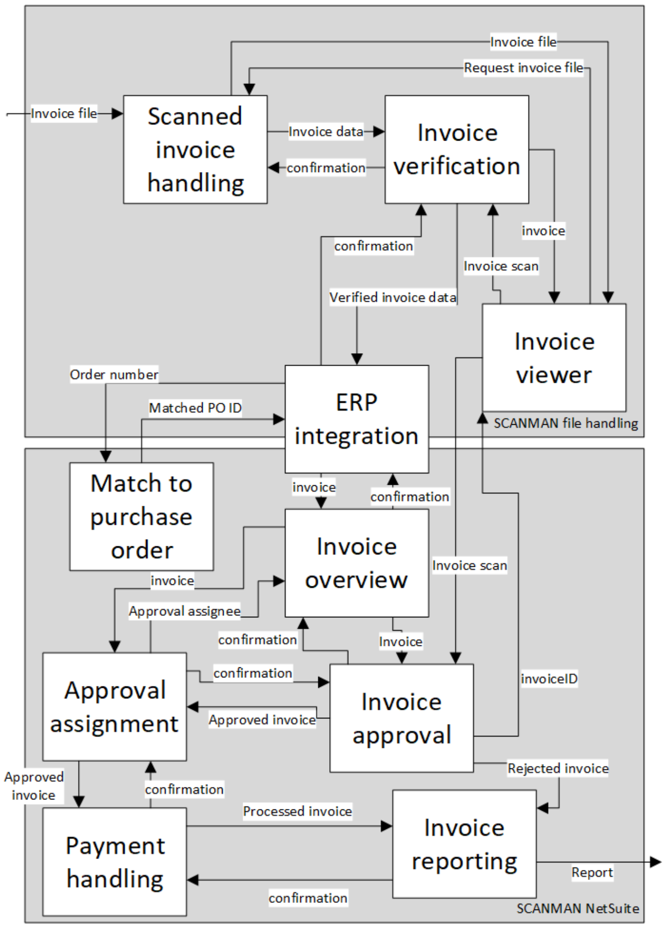

Fig. 7. FAD of the SCANMAN NetSuite intended architecture

The design for a first version of the NetSuite add-on consisted of a total of 10 ES, and 70 US. This resulted in 10 modules and 69 features. The reason why we have almost a 1:1 relation between the RE and SA concepts is that this case study is based on the intended architecture, instead of the actual architecture (like in the other 3 cases). The reason for not using the actual architecture is that the project is still in progress as of the time of writing. This does indicate that the RE4SA model can be applied to transform the RE artefacts to an intended architecture. The developer can then refine this intended architecture with the 'common sense' features, and the developer's own additions or alternative solutions to the requirements. Afterwards the intended architecture needs to be updated to reflect the actual architecture.

The initial design for the NetSuite version only has about 5% of features compared to the JDE version. This can be explained, as the initial design only consists of the must haves of the application. It also describes the intended architecture (Fig. 6) as opposed to the actual architecture (based on the GUI) which will most likely have more features. Finally, the difference could be that the software architecture is created before the application, which would result in a more efficient solution.

E. *Case study metrics*

To obtain quantitative insights on the absolute and relative frequency of the RE4SA artefacts, we noted all the values and calculated the ratios between the connected artefacts. These can be seen in Table II; the numbers indicate that there is not necessarily a 1:1 relation between the RE and SA concepts. There are cases where we have more US than features, and the other way around. Modules and epic stories are closer to the 1:1 relation with only one exception. However, the lower number of ES makes the sample size smaller and therefore the results less generalizable. The ES : US ratio can differ from the total ES divided by total US, because there are US that are linked to existing modules instead of referring to the elements specified in the ES. The average number of user stories per epic in the cases was 7.6, while the average number of features per module 20.3.

Table II. Metrics for the RE4SA artefacts in the four cases

Case	**1**	**2**	**3**	**4**
Frequencies				
User stories (US)	32	14	40	70
Features	21	28	104	69
Epic stories (ES)	0	1	3	10
Modules	0	1	4	10
Ratios				
US : Features	1.5: 1	1 : 2	1: 2.6	1 : 1
ES : US	N/A	1: 7	1: 8	1: 7
ES: Modules	N/A	1: 1	1: 1.3	1: 1
Modules: Features	N/A	1: 28	1: 26	1: 6.9

We also analysed the complete feature diagram of the application after the update described in case 3. For this the metrics are based on the Quality in Use Integrated Measurement (QUIM) model by Seffah et al. [27] These metrics can help determine usability aspects of GUI-style applications. The results of this analysis can be seen in Table III. We include the table for use in future research to determine usability aspects of an application based on the feature diagram.

V. Case Study Evaluation & Lessons Learned

A. *Expert interviews*

The four cases were evaluated through interviews with experts involved with the project of the case studies. These experts play different roles within the teams and company; developers, functional consultants, project managers and JDE deployment experts were interviewed. In each of the evaluations, a semi-structured interview was used, the experts were first presented with the results of the case study, and then asked ten questions to obtain their opinion on the cases and application of the RE4SA model. In this section we will discuss the findings of these interviews.

Table III. Feature diagram metrics

Metric	**Value**
Number of modules	11.0
Number of submodules	55.0
Average atomic features per (sub) module	20.6
Average degree	9.2
Max degree	323.0
Min degree	1.0
Average standard deviation of degree	9.8
Average degree without grids	5.0
Average standard deviation of degree without grids	2.2
Average depth	2.1
Max depth	6.0
Min depth	1.0
Average standard deviation of depth	0.4

All the experts said they had limited knowledge and experience with user stories, since the company does not currently use them. However, user stories were also introduced through a second source: a new partnership. Only one of the experts had worked with user stories before. However, even with this limited knowledge, opinions were very positive; all interviewees felt that the RE4SA concepts adequately captured the customers' requirements, that they provide a detailed overview, and could effectively be applied for changes in requirements. Only one interviewee doubted if they contain enough information for a developer. The other interviewees did think the concepts gave enough information but were unsure if it would work properly if development was completely outsourced, as this often requires stricter guidelines since outsourced workers lack context information for projects. It was mentioned by multiple interviewees that a big factor for adoption are the skills and experience of the developer. The interviewed developers all thought the documentation would contain enough information for development, however, one of the developers mentioned that not all requirements can be captured before starting development.

All interviewees were positive about the software architecture diagrams, but their expected use cases for the diagrams differed. Most experts said the diagrams gave a clear overview of the software and thought that it would be useful for new employees to learn about the application. Other use cases that were mentioned are: as a checklist, for testing purposes, detecting dependencies with other elements.

When asked if the SA diagrams would be used for communication with customers, the opinions were less uniform: some interviewees said they would use (parts of) the SA in communication to show the way the application works, ensure they are talking about the same aspects, or ensure that the features fulfil requirements. It was also mentioned that the SA diagrams could be used to visualise and communicate the risks of a requested customisation. The other interviewees thought the diagrams would be too complicated for use in customer communications, the feature diagram would be only useful in this scenario if cut-outs were used as opposed to the full diagram.

When presented with the scenario where customer-specific customisations were colour coded, all experts agreed on its usefulness. One of the interviewees mentioned that every time the customers' environment was accessed, it was unknown what changes had been made in that environment. And that having a SA on a customer level (extending a template from the standard version) would be a time saver.

While all experts thought that applying RE4SA would be an improvement over the current method, they also acknowledged some disadvantages. Three of them mentioned that while the RE4SA concepts provide a very detailed overview of the application, they would quickly become useless if they are not kept up to date. Due to the size of the application, the feature diagram can be overwhelming which could limit the acceptance for using it. It would also increase time spent on documentation, frontloading more effort but (hopefully) reducing the need for rework and improving available documentation. Finally, it could be that developers feel limited due to the specific solutions mapped in the feature diagram of the intended architecture. This could be mitigated by communicating clearly that the initial feature diagram is only a suggestion.

B. *Lessons learned*

From execution and analysis of the case studies we gained some additional insights in the use of the RE4SA model. The most important lessons that we've learned from performing the multi-case study are presented in. The lessons have been grouped based on the RE4SA concepts they refer to.

VI. Discussion and future research

In this paper, we have conducted a multi-case study where we applied the RE4SA model to a customisation scenario of enterprise software. We have shown that the RE4SA model can be applied to link requirements to the software architecture and ensure that these match with one another. By updating the SA based on new requirements, we can ensure alignment between the requirements and architecture. The situational method enables the company to keep an up to date architecture of the software product in a way that minimizes required effort.

We have applied the method to each of the cases and received positive expert feedback. Through this multi-case study, we have provided evidence for the use of the RE4SA model in Enterprise Software customisation and management, thereby addressing our RQ. Application of the RE4SA model improved the design process by increasing the goal focus of requirements, providing a clear overview of customer specific environments, and providing an overview of the location of features and modules in a new software version. Colour coding was applied in the first three cases and allows us to emphasise certain features or models. This was especially useful to visualize customisations and new features. Also, applying the model results in concise yet detailed documentation, which could be possibly used to improve the communication with customers.

Table IV: Lessons learned from performing the case studies

	User stories & features
1	**User stories leading, feature diagram as an aid.** It needs to be stressed in communication with developers that the user stories are the main focus and that the feature diagram is only an aid. Otherwise the developers will feel restricted, and they may lose the focus on goal orientation. This does mean that the feature diagram needs to be updated to reflect the actual architecture after development
2	**Tracing requirements to features.** User stories from a customer can be traced to features in the feature diagram to show that their requirement is already met (and how). This can be assisted through concept recognition, where certain terms in the requirements can be linked to a part of the feature diagram, or functional architecture diagram.
3	**Placing new features.** New features could be placed in a logical location for the software by first considering the different modules that could contain the new feature, and then looking at existing features within those modules to see if any are similar to the new features. Candidate modules can be identified by looking at the concepts in the user story.
4	**Feature & user story metrics.** The cardinality and ratios of features and user stories differ. In our study, we found a feature described by many user stories, e.g., when different roles want the same thing. In other cases, multiple features were combined to meet one user story.
5	**Undocumented features.** Not all features are described in user stories; sometimes, this occurs because some features are common sense and defining user stories for these would be redundant. For example, when a user story defines a customer profile, it can be deemed unnecessary to create a user story to describe the need for a username. Alternatively, a developer might add features that are not described in a user story because he thinks it is important.
	Features
6	**Customer specific feature diagrams.** Copies of a feature model can be easily created using software tools like the Eclipse modelling tool [28], thereby reducing the effort for creating customer-specific feature diagrams. These-customer-specific environments provide a lot of value, as they efficiently document customisations.
7	**Feature diagram comprehension.** As feature diagrams grow in size, they become harder to comprehend which limits their use in communication. This can be mitigated by using only the relevant part of the feature diagram so that only the needed information is shared.
	Feature & modules
8	**Colour coding features or modules.** Colour coding can guide the viewer to relevant features or modules. Colour coding is especially useful to visualise aspects that deviate for an existing software product (e.g., customised or new). Colour coding can also partly solve the issue that the feature diagram becomes incomprehensible due to its size.
9	**Iterative SA refinement.** Applying the situational RE4SA method in multiple projects for a software product allows for refinement of the software architecture models. This means over time the accuracy of the model compared to the actual implementation improves.
10	**Consistency in functional SA documentation.** By creating/updating the feature diagram and functional architecture diagram in parallel the models can be made more consistent and be tested for completeness. This includes ensuring elements have the same names in different models and are at the same level of depth.
11	**Update intended to actual architecture.** The feature diagram should be updated after development to properly reflect the actual architecture as opposed to the intended architecture.
12	**Drill-down navigation.** In order to properly support the information seeking mantra [23] through drill-down navigation, the different functional architecture views should be linked to each other. Tool support is essential to do so.
13	**Zoom out navigation.** While the drill-down navigation might be the most common use case for functional experts, developers might benefit more from being able to zoom-out starting from the feature level. This would allow them to obtain an overview of the functionalities instead of only working on the low-level details.
14	**Version update visualisation.** Added features and modules can be modelled in the FAD and feature diagrams. By colour coding the new elements, it can easily be seen which additions were made in the update. This can be used to create the release notes, and to identify in which update a specific feature / module has been added.
	User stories, epic stories, features & modules
15	**Identification of new features / modules in SA creation.** During the creation of the software architecture new features and modules can be identified, by using the link between software architecture and requirements, requirements can be recovered based on the architecture.
16	**Barista problem.** Using the proper level of granularity for US and ES remains a difficult task. With RE4SA, this problem also extends to the SA level: what is a composite feature or a sub-module? However, since each of the diagrams is a different view of the same information, it should not be an issue if certain aspects are contained in multiple views. These views can be added when they are needed.
17	**Concise yet detailed documentation.** The resulting documentation was perceived as more detailed by the experts/interviewees, whilst decreasing the amount of text required. By utilizing the requirements for documentation purposes, we can keep track of the “who, what and why” of a solution, while the SA keeps track of the “how”.
18	**Facilitate communication between functional experts and developers.** As a result of lessons 12 & 13, we can support a shared context knowledge between developers and functional experts. As developers can zoom out from their normal focus on the feature level, and functional experts can zoom in from their normal focus on module level.
19	**Customisation risk assessment.** By utilizing the software architecture models, the risk of a requested customisation can be assessed. This can be done by analysing the information requirements from the modules, and the hierarchy of the suggested customisation in the feature diagram.

We also obtained initial evidence for some of the hypotheses by Molenaar *et al.* [12]. Case 1 shows that the RE4SA model can support placing new functionalities in an existing software. The identified use cases for the SA aspects in the expert evaluation also provide evidence for the hypothesis that the model can be used to guide and support testing activities. Finally, the version update case shows that the model can potentially be used for release planning.

In this research, the hypothesis [12] that there would be a 1:1 relationship between USs and features seems not to hold true. This can be explained as multiple roles might require a specific feature, which would result in multiple USs per feature. Multiple features could also solve a single US as seen in Fig. 6. For the cardinality between ES and modules our findings were more consistent with the expected cardinality, as in two cases there was a 1:1 relationship and in one there was a 1:1.25 relationship.

A. *Validity threats*

We have tried to minimalize the validity threats by following established case study guidelines [13], building on previous research [12, 19, 26], and performing four case studies, which should increase the precision due to data triangulation [23]. However, as with all case studies, there are validity threats that limit the generalizability of our research. All of the cases were focused on a single enterprise software application and focus on a single company. Forza IT group provided all available documentation, and time of experts which facilitated the research; however, since the research was done in combination with an internship, this might have introduced a bias. The interviewees all knew the interviewer, and that they were part of a study, which might have led to more positive reactions. As mentioned in the expert interviews section, the experts had limited experience with user stories. This could cause validity threats as they might have a more favourable opinion because they were shown 'something new'. This is somewhat limited as it has also been introduced from a second source aside from this research (the partnership). Furthermore, the effectiveness of user stories has been proven in previous research, and by the high adoption numbers. It is also possible that due to general human resistance to change, the reactions were less positive than if they had been familiar with US. Finally, as Forza did not have a completely structured method for their projects, they may have provided more positive interview reactions since any structured method might be an improvement.

B. *Future research*

While we have obtained some evidence that the RE4SA can be applied effectively in this case study, it would be beneficial to conduct additional case studies to obtain more evidence for the findings from this research. One particularly interesting aspect concerns the effectiveness of the RE4SA artefacts regarding communication with customers, as the findings in this research were all based on views from practitioners within the product team. Furthermore, our scope was mostly on the design phase; this research could be continued by applying the model with a higher focus on the requirements elicitation and development phases.

For outsourcing development, designing the solution with the RE4SA model and enforcing the creation of features as described in the feature diagram might lead to effective results. Outsourcing development often has the issue that the developers lack context information of the application and possibly end up creating the wrong solutions. Further research on application of the RE4SA model on outsourced development could provide proof for this hypothesis.

We also identified a possibility to support the RE4SA model through software tools that allow for establishing and maintaining traceability. By linking all artefacts through a tool, the amount of manual work required to use the method can be minimized and the trace links between the RE and SA disciplines can be fully utilized. Utilizing natural language processing it might also facilitate the automated creation of the SA based on a requirements set. There is ongoing research on the creation of these tools, and Lucassen et al. [29] have established traceability between user stories and source code.

Finally, aiming to reduce the effort required to create RE and SA documentation, we would like to investigate the automation of requirements reporting through speech and action recognition, similar to the research on automation of medical reporting in Care2Report [30].

Acknowledgment

We would like to thank the GRIMM research group at Utrecht University for their feedback and comments throughout the research. We would also like to thank Forza IT group for their assistance, input on the research, and access to artefacts and experts.

References

[1] A. Aurum and C. Wohlin, "Requirements Engineering: Setting the Context", in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Red. Berlin/Heidelberg: Springer-Verlag, 2005, pp. 1–15.

[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] D. M. Fernández e.a., "Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice", Empirical Software Engineering, vol. 22, nr. 5, pp. 2298–2338, okt. 2017.

[4] P. Clements, D. Garlan, R. Little, R. Nord, and J. Stafford, "Documenting software architectures: views and beyond", in 25th International Conference on Software Engineering, 2003. Proceedings., Portland, OR, USA, 2003, pp. 740–741.

[5] N. Rozanski and E. Woods, Software systems architecture: working with stakeholders using viewpoints and perspectives. Upper Saddle River, NJ: Addison-Wesley, 2012.

[6] M. Lindvall and D. Muthig, "Bridging the Software Architecture Gap", Computer, vol. 41, nr. 6, pp. 98–101, jun. 2008.

[7] B. Nuseibeh, "Weaving together requirements and architectures", Computer, vol. 34, nr. 3, pp. 115–119, mrt. 2001.

[8] G. Lucassen, F. Dalpiaz, J. M. van der Werf, and S. Brinkkemper, "Bridging the Twin Peaks -- The Case of the Software Industry", in 2015 IEEE/ACM 5th International

Workshop on the Twin Peaks of Requirements and Architecture, Florence, Italy, 2015, pp. 24–28.

[9] I. Shin, ‘Adoption of Enterprise Application Software and Firm Performance’, Small Business Economics, vol. 26, no. 3, pp. 241–256, Apr. 2006.

[10] Z. Zhang, M. K. O. Lee, P. Huang, L. Zhang, and X. Huang, ‘A framework of ERP systems implementation success in China: An empirical study’, International Journal of Production Economics, vol. 98, no. 1, pp. 56–80, Oct. 2005.

[11] Panorama, “Biggest ERP Customization Challenges.” 2014 ERP report retrieved from: https://www.panorama-consulting.com/tuesday-poll-erp-customizations/

[12] S. Molenaar, S. Brinkkemper, A. Menkveld, T. Smudde, R. Blessinga, F. Dalpiaz. “On the Nature of Links between Requirements and Architectures: Case Studies on User Story Utilization in Agile Development.” Technical report UU-CS-2019-008. Department of Information and Computing Sciences, Utrecht University, the Netherlands. June 2019.

[13] I. van de Weerd and S. Brinkkemper, ‘Meta-modeling for situational analysis and design methods’, in Handbook of research on modern systems analysis and design technologies and applications, IGI Global, 2009, pp. 35–54.

[14] S. A. Fricker, ‘Software Product Management’, in Software for People, A. Maedche, A. Botzenhardt, and L. Neer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–81.

[15] S. R. Palmer and M. Felsing, A Practical Guide to Feature-Driven Development, 1st ed. Pearson Education, 2001.

[16] R. L. Nord and J. E. Tomayko, ‘Software architecture-centric methods and agile development’, IEEE Software, vol. 23, no. 2, pp. 47–53, Mar. 2006.

[17] M. Kassab, ‘The changing landscape of requirements engineering practices over the past decade’, in 2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE), Ottawa, ON, Canada, 2015, pp. 1–8.

[18] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, ‘The Use and Effectiveness of User Stories in Practice’, in Requirements Engineering: Foundation for Software Quality, vol. 9619, M. Daneva and O. Pastor, Eds. Cham: Springer International Publishing, 2016, pp. 205–222.

[19] 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: Cham: Springer International Publishing, 2018, pp. 235-252.

[20] S. Brinkkemper and S. Pachidi, ‘Functional Architecture Modeling for the Software Product Industry’, in Software Architecture, vol. 6285, M. A. Babar and I. Gorton, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 198–213.

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

[22] A. Klement, ‘replacing the user story with the job story’, https://jtbd.info/replacing-the-user-story-with-the-job-story-af7cdee10c27 , Nov. 2003

[23] P. Runeson and M. Höst, ‘Guidelines for conducting and reporting case study research in software engineering’, Empirical Software Engineering, vol. 14, no. 2, pp. 131–164, Apr. 2009.

[24] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, ‘Feature-Oriented Domain Analysis (FODA) Feasibility Study’:, Defense Technical Information Center, Fort Belvoir, VA, Nov. 1990.

[25] D. Traynor, ‘Intercom on Jobs-to-be-done.’ Book retrieved from interface.com

[26] B. Shneiderman, ‘A Task by Data Type Taxonomy for Information Visualizations’. In The craft of information visualization. Morgan Kaufmann, 2003, pp. 364-371

[27] Seffah, A., Donyaee, M., Kline, R. B., & Padda, H. K. “Usability measurement and metrics: A consolidated model.” Software quality journal, 14(2), 2006, pp. 159-178.

[28] Eclipse modelling tool , 2019 https://www.eclipse.org/downloads/packages/release/kepler/sr2/eclipse-modeling-tools

[29] Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., Brinkkemper, S., & Zowghi, D. . Behavior-driven requirements traceability via automated acceptance tests. In 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW) (pp. 431-434). IEEE.

[30] Care2Report, Automated medical reporting. https://sites.google.com/view/care2report