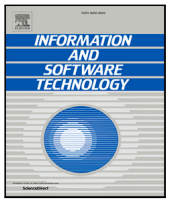
Information and Software Technology 133 (2021) 106535

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Alignment and granularity of requirements and architecture in agile development: A functional perspective

Tjerk Spijkman [a,b,*], Sabine Molenaar [a], Fabiano Dalpiaz [a], Sjaak Brinkkemper [a]

[a] Utrecht University, The Netherlands
[b] fizor., The Netherlands

ARTICLE INFO

Keywords:
Requirements engineering
Software architecture
Twin Peaks
Alignment
Granularity
Case study
Agile development

ABSTRACT

Context: Requirements engineering and software architecture are tightly linked disciplines. The Twin Peaks model suggests that requirements and architectural components should stay aligned while the system is designed and as the level of detail increases. Unfortunately, this is hardly the case in practical settings.
Objective: We surmise that a reason for the absence of conjoint evolution is that existing models, such as the Twin Peaks, do not provide concrete guidance for practitioners. We propose the Requirements Engineering for Software Architecture (RE4SA) model to assist in analyzing the alignment and the granularity of functional requirements and architectural components.
Methods: After detailing the RE4SA model in notation-independent terms, we propose a concrete instance, called RE4SA-Agile, that connects common artifacts in agile development, such as user stories and features. We introduce metrics that measure the alignment between the requirements and architecture, and we define granularity smells to pinpoint situation in which the granularity of one high-level requirement or high-level component is not uniform with the norm. We show two applications of RE4SA-Agile, including the use of the metrics, to real-world case studies.
Results: Our applications of RE4SA-Agile, which were discussed with representatives from the development teams, prove to be able to pinpoint problematic situations regarding the relationship between functional requirements and architecture.
Conclusion: RE4SA and its metrics can be seen as a first attempt to provide a concrete approach for supporting the application of the Twin Peaks model. We expect future research to apply our metrics to additional cases and to provide variants for RE4SA that support different concrete notations, and extend the perspective beyond the functional perspective of this research, similar to what we did with RE4SA-Agile in this paper.

1. Introduction

Requirements engineering (RE) and software architecture (SA) are entangled disciplines. Nuseibeh's Twin Peaks model [1] describes how requirements and architecture undergo conjoint evolution: while they are separate activities, the former guides the latter and the latter constrains the former. The relevance of the Twin Peaks has been clearly acknowledged by the research community [2] and open challenges exist [3] including communication, preserving architectural knowledge, and reconstructing requirements. Previous work has also proposed adaptations of the Twin Peaks, e.g., for agile development of software products [4] and for product lines with variability [5].

Our work focuses on the role of artifacts in particular, functional requirements and functional architectural models to support the collaboration within and across disciplines. Researchers have pointed out how software engineering is a social activity among humans [6] and have shown the key role of communication among the various domains of software engineering [7].

Communication problems affect both RE and SA. In RE, flawed communication is a prevalent cause of project failure [8]. This is exacerbated by the fact that user and client needs change continuously, leading to volatile requirements [9,10]. While written artifacts are common in RE, SA suffers from the lack of proper documentation, leading to heightened risks of architectural drift and erosion, as well as increased costs and a decrease in software quality [11]. Finally, changes in requirements have been shown to endanger component reuse [12].

In this paper, we support improved communication between RE and SA by providing concrete guidance for the conjoint evolution of requirements and architecture. While previous works identified challenges in

* Corresponding author at: Utrecht University, The Netherlands.
E-mail address: tjerk@fizor.io (T. Spijkman).

https://doi.org/10.1016/j.infsof.2021.106535
Received 15 July 2020; Received in revised form 16 January 2021; Accepted 18 January 2021
Available online 29 January 2021

applying the Twin Peaks model [1,3,4], they did not specify *how* to tackle them. The main research question (MRQ) of our research is as follows:

MRQ. *How to assist agile development teams in applying the Twin Peaks model?*

While the Twin Peaks model concerns both functional and non-functional aspects [3], we focus here only on functional requirements and functional architecture, which are hard to align in the context of software products [4]. We leave the study of the even more intricate non-functional perspective to future research.

The Twin Peaks model considers a vertical dimension (within RE and within SA) and an horizontal dimension (across RE and SA). We study both dimensions through the notions of granularity and alignment, respectively. Thus, we divide the MRQ into two research questions (RQs) regarding the two dimensions:

RQ1. *How to assist in achieving uniform **granularity** within and between functional requirements and architecture specifications?*

RQ2. *How to assist in establishing and maintaining **alignment** between functional requirements and architecture specifications?*

In this research, we focus on functional requirements within the context of software products. This type of requirement is particularly relevant because new functional requirements emerge to adapt a product to specific customers, to reap technological opportunities, and to realize the product strategy [13]. Future work should complement our perspective with the study of quality requirements, which are cross-cutting concerns that spread across multiple components[1] [14,15]. While we acknowledge that there are multiple views on software architecture, our focus is on providing techniques for reconciling the artifacts from RE and SA, rather than on the processes involved in the creation of those artifacts.

We present the Requirements Engineering for Software Architecture (RE4SA) model that links functional requirements and architectural components. Fig. 1 overlays the RE4SA concepts over the Twin Peaks model [1]; the conjoint evolution is represented by the spiral arrow that *aligns* artifacts from both disciplines[2] while increasing the *granularity* from high-level to detailed. In an effective application of RE4SA, high level requirements are aligned with high level architecture, just like detailed requirements are aligned with detailed architecture.

We borrow the definition of software architecture by Bass et al. [16]: "The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both". As mentioned in earlier paragraphs, we particularly focus on the artifacts that document the functional view of the software. While the components of a software architecture can be *justified* by multiple requirements, we assume here that a functional requirement mainly *describes* a single functional component.

In this research, we build on our earlier work [17]. More specifically, we extend the set of metrics to cover the granularity dimension of the model. Additionally, we position a general version for our instance of RE4SA, and we revise and extend the case studies and literature sections. Finally, we add running examples to the paper to illustrate the concepts that we introduce.

[1] In this research, we refer to components in the general definition of the word: "A part or element of a larger whole". While we are aware there are different connotations within software architecture, we feel this best conveys our message.

[2] We use the term *alignment* instead of *implementation dependence* (from the Twin Peaks model) to better emphasize the fact that the architecture may lead to new needs that were not indicated in the requirements.

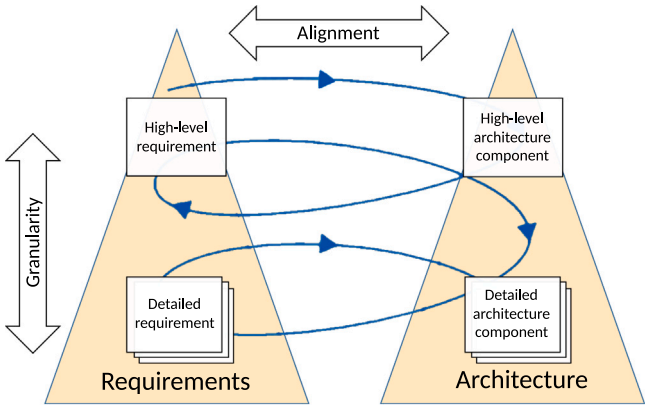


Fig. 1. RE4SA — Twin Peaks overlay.

The RE4SA model is intended as a guideline to applying the Twin Peaks model and a means to facilitate communication and system specification. While this solution requires some upfront work, aimed at creating or recovering the architecture and linking the requirements, we expect it to decrease rework in the subsequent development phase. Specifically, we make the following contributions:

- We present the RE4SA general model for linking RE and SA artifacts, and one concrete instance for agile development called RE4SA-Agile [17,18], which includes user stories, among other notations.
- We introduce *granularity smells and metrics* as a mechanism to pinpoint RE or SA artifacts that are excessively coarse-grained or excessively fine-grained compared to the norm. The thresholds for identifying these smells have been determined empirically through the analysis of eleven real-world requirements datasets.
- We introduce metrics for analyzing in a quantitative manner the alignment degree between RE and SA artifacts.
- We report on two case studies that apply RE4SA-Agile for the purpose of architecture discovery and architecture recovery, respectively. We analyze the data both in terms of granularity and alignment.

The datasets used in this paper, which represent RE and SA artifacts, are made publicly available for transparency and to promote their reusability.[3]

Research approach. Our approach is predominantly empirical. We answer RQ1 and RQ2 (which are "how" questions) by proposing multiple design artifacts: (i) the RE4SA and the RE4SA-Agile models that represents the artifacts in the RE and SA domains and the relationships therein; and (ii) metrics that allow measuring the degree of granularity (RQ1) and alignment (RQ2). In particular, based on industry standards and well-known concepts, the RE4SA model for agile development has been previously proposed and applied it using field study to two real-world cases without interference [17]. This version is called RE4SA-Agile here. We introduce the RE4SA general model and we conduct theory building by formulating metrics that can be used to assess the granularity (RQ1) and alignment (RQ2) of artifacts covered by the model. We define specific hypothesis (H1 and H2 in Section 6) that allow us to conduct a first validation of the metrics through an application to the real-world cases, leading to additional findings.

[3] The dataset can be found at https://sandbox.zenodo.org/record/679359 or found through DOI http://dx.doi.org/10.5072/zenodo.679359.

Organization. Section 2 discusses background work. In Section 3, we present the RE4SA model and RE4SA-Agile specification, followed by the granularity metrics in Section 4 and alignment metrics in Section 5. Section 6 illustrates how the RE4SA-Agile model and its metrics can be applied in practice, using two case studies. A discussion on the research and validity threats can be found in Section 7, followed by the conclusion and future research in Section 8.

2. Background: Granularity and Alignment of RE and SA artifacts

Agile development methods have changed how software is created, development is more iterative and a focus shift from documentation to communication [19]. This in turn has impacted and created challenges for the RE and SA domains as the context for their activities has changed. Instead of detailed system specifications, requirements are documented in formats better suited to agile development, like user stories, prototyping, and scenarios [20]. Agile requires incremental construction of a product's functionality, this calls for a modular architecture with minimal coordination with other modules and easy to extend [21].

In Section 2.1, we first discuss research related to the Twin Peaks model to provide an overview of similar approaches in the relationship between RE and SA. After that, we review literature that is relevant to the two relationships as seen in Fig. 1: granularity and alignment, in Section 2.2 and Section 2.3, respectively.

2.1. Twin peaks

The Twin Peaks model [1] details the tight relationship between RE and SA, and supports the iterative specification of both the requirements and architecture suited to Agile. Rashid et al. [22] focus on identification of conflicting aspects and determining trade-offs in the requirements before deriving the architecture. However, their approach is not based on iterative development. Ameller et al. [23] performed an empirical study on how non-functional requirements impact architecture design in practice. Whalen et al. [24] build on the Twin Peaks mindset and argue for co-evolution of RE and SA in hierarchical systems, stating that these are often designed on a middle out approach on granularity, both abstracting and refining the requirements and architecture. The COSMOD-RE [25] method positions a co-design approach for RE and SA, by applying requirements and architecture viewpoints to four layers of abstraction for system design. Similarly, the CBSP approach is an approach that can be applied to design an architecture from a set of requirements [26]. Brandozzi and Perry investigated a specification language to derive architecture and state constraining requirements [27,28]. The Global analysis activities and artifacts described by Hofmeister et al. [29] serve to reduce the gap between RE and SA. Van Lamsweerde [30] positions an approach that considers the relation between RE and SA when designing alternative ways to cope with a requirement. In their research, when alternative ways to achieve goals in goal-oriented RE are considered, the impact on architecture is considered to constrain the requirements and determine the best way to evolve the software. Hall et al. [31] extend the research on problem frames [32] and bring it within the context of the Twin Peaks model. The reciprocal Twin Peaks model [4] extends the Twin Peaks model by specifying how the domains can be linked by considering the responsibilities in both and further links the model to agile development.

2.2. Granularity

From a functional perspective, software can be decomposed into components that contain different functionalities and have their own responsibilities [33]. This decomposition is commonly referred to as a module(-based) structure [16,34]. Modules should contain functional responsibilities, which are divided based on the principle of separation of concerns. Such modules are broken down into smaller so-called submodules until they are small enough to be understood [16]. Submodules should have no overlapping responsibilities, but should jointly contain all the responsibilities of the module they are a part of [34].

In more detail, software requires capabilities to implement features that are expected of application in a certain domain. We take the definition of feature by Apel and Kästner [35]: "*a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option*". We acknowledge that architecture granularity can encompass more than the functional perspective discussed in this section, for example, the quality and performance of a solution. However, we do not cover this in the related works as these surpass the scope of the research.

Decomposing a complex system into discrete parts that can communicate with each other allows for a more manageable representation of the system, this process is also referred to as modularization [36]. There is no single approach to modularization or decomposition. However, since the nature of modularity is the same across software engineering activities, we can learn from examples set by other fields. One such example is Business Process Management, which applies modularity by decomposing processes into subprocesses. Davis [37] states there is no objective approach to determining the correct level of granularity. To that extent, it cannot be said with full certainty whether a subprocess should be on the lowest level of granularity or the one above. Instead, consistency in the levels of granularity is key. Reijers et al. [38] describe three criteria to determine whether nodes should be included in the same subprocess or in separate subprocesses: (i) block-structuredness, (ii) connectedness, and (iii) similarity of labels. Firstly, block-structuredness refers to whether the process has a single entrance and a single exit. Arguably, this criterion can be applied to modules. However, the input and output flows of modules are less apparent and the size of the modules should be taken into account to avoid elements that have disproportionately high amount of functionality [14]. Secondly, scenario overlays [39] can be utilized to determine the connectedness of features (nodes) within modules (subprocesses). Thirdly, the similarity of labels can applied to both the requirements and architecture. Requirements and architecture components can be grouped together based on commonalities in the artifacts. For instance, a feature called `login to system` is more likely to belong to the same module as the `recover password` feature, than the `request travel expense` feature.

None of these criteria, however, give any indication on the number of elements to include in a group. Metrics defined by e Abreu and Goulão [40] provide guidance on how many classes a module should contain in relation to Object-Oriented systems. They base the (relative) size of modules on the relative dispersion of classes, to mitigate the effect of modules with skewed distributions. They divide the total number of classes by the total number of modules, this result then divides the difference between the highest number of classes contained in a module and the lowest number of classes, to determine the relative module dispersion.

Even if a requirement has good quality features, it can still be negatively impacted by its granularity. A flaw in the granularity can mean that a user story is either too concrete, or too abstract in the system scope [41]. This could, for example, lead to user stories that cause difficulty in determining the effort estimate, are difficult to implement as they are too abstract, or limit the developer if they are too strict. Liskin et al. [41] suggest assessing the granularity of a user story based on the expected implementation duration. España et al. [42] note that an analyst relies on methodological guidelines to encapsulate concepts. They use unity criteria to determine the granularity of encapsulation, and identify two type of granularity errors: (i) functional fragmentation error, when two or more encapsulations should have been modeled as a single encapsulation, and (ii) functional aggregation error, when one functional encapsulation should have been modeled as two or more encapsulations according to the unity criteria. Kästner et al. [43]

discuss similar difficulties in granularity for feature implementation. Differences in granularity can make it difficult to understand and maintain modularization of features. They found, on closer inspection of their previous projects, that they had unnecessary replication of code due to granularity issues. They built an eclipse based prototype to decompose an application into features with fine granularity and use background colors to show differences in granularity. They do however, take a code-focused approach, as opposed to our focus on functional architecture. Through the link between the functional requirements and architecture utilized in RE4SA, we intend to advance the research on granularity and introduce indicators for granularity issues in a set of requirements or architectural components.

2.3. Alignment

As McKeen and Smith [44] argue that alignment between business and IT is a state in which the goals and activities of a business are in harmony with the information systems that support them, we apply the same line of thought to alignment between requirements and architecture. Thus, alignment between requirements and architecture is a state in which the requirements specification is in harmony with the architectural specification and both describe the same application. Perfect alignment between requirements and functional architecture is a state in which all the system requirements are satisfied by a component in the architecture, and all components in the architecture can be linked to the requirements. Keeping software artifacts aligned falls under the umbrella term of software traceability [45], which includes techniques for establishing and maintaining trace links between different artifacts like requirements, architecture, code, and tests. Among the open challenges that pertain to our work, *ubiquitous traceability* [46] is especially important, as it stresses the need of tools and techniques that minimize the required human effort to create and keep the trace links up to date.

Many automated tools exist for the automated establishment of trace links. Trace Analyzer [47] uses certain or hypothesized dependencies between artifacts and common ground and then considers nodes that contain overlapping common ground to establish a trace link. The common ground they use, however, is source code, which is unusable when the system is still under design. Zhang et al. [48] use an ontology-based approach to recover trace links, but only link the source code to documentation. Traceability links have also been explored in agile development, with a focus on establishing links between commits and issues [49].

The systematic mapping by Borg et al. [50] shows that the most frequently studied links in information retrieval-based traceability are the links between requirements and between requirements and source code. Other popular links are between requirements and tests, and other artifacts and code. Linking requirements and architectures is a less studied topic.

Tang et al. [51] study the creation of traces between requirements and architecture. They provide an ontology for annotating manually specifications and architectural artifacts, which are then documented in a semantic wiki. This wiki shows which architectural design outcome realizes which requirement, which decisions have been made, and the links to quality requirements.

Rempel and Mäder [52] are among the first ones to propose traceability metrics in the context of agile development. They propose graph-based metrics that link requirements and test cases. Numerous researchers in the field of software maintenance proposed metrics, starting from the seminal work by Pfleeger and Bohner [53]. Our work, however, focuses solely on metrics between requirements and architectures in the context of agile development for software products.

Recently, Murugesan et al. [54] presented a hierarchical reference model to capture the relationship between requirements and architecture. Their goals are similar to those of this research, but they focused on *technical* architectures. Our work, instead, investigates *functional* architectures and suggests the use of specific artifacts to formulate more specific guidelines, as opposed to a generally applicable requirement-to-component connection model.

3. The RE4SA model

In this research, we focus on the relationships between the artifacts in both the RE and SA domains. We aim to provide metrics to measure the alignment between the artifacts, to facilitate communication within a development team, and to detect architecture or requirements smells [55]. We propose the Requirements Engineering for Software Architecture (RE4SA) model (Fig. 2(a)), and an instance of this model that includes concrete notations, assembled based on tight collaboration with industrial partners in the software domain (Fig. 2(b)).

Like the Twin Peaks model, RE4SA links the RE and SA domains. More specifically, it relates the problem space, which describes the intended behavior through requirements, to the solution space that defines how such intended behavior is implemented, i.e., how the requirements are satisfied [35]. This connection is considered on two different levels of granularity. In practice, requirements are often grouped to denote a similar goal or scope; for example, via themes, epics, or use cases. In agile development, these groupings are often used to determine the scope of a sprint or release, as they indicate a shared functionality [56,57]. This is likely to lead to a similar grouping in the architecture, since the requirements form the basis for design and development.

In this paper, we generalize the definition of the RE4SA model that we presented in our previous publications [17,18]. What we called RE4SA is now renamed to RE4SA-Agile, which is an instance of the general model (Fig. 2a) referred to as RE4SA. While the proposed RE4SA model is kept generic to allow for its use with different artifacts, the following sections rely on some assumptions/limitations: (i) detailed components should belong to at most one high-level component; (ii) the granularity levels between the RE and SA concepts need to be at a similar level. We envision the use of RE4SA as a lens for researchers to explore the vertical and horizontal relationship within and across the RE and SA disciplines, without being bound to the notations that we include in RE4SA-Agile.

To ease explanation and illustration, we limit our focus on two levels of granularity. In the RE4SA model, the concepts are given general names, but in practice there could be multiple levels of these concepts. For example, high-level 1 and high-level 2 aspects. In scenarios with multiple layers, the relationships between the concepts should be applied to two adjacent layers in the model.

Furthermore, the relationships between the concepts can be classified depending on whether they affect the *granularity* of the specification (refinement and abstraction) or they support the *alignment* between requirements and architecture components (allocation and satisfaction). While we want to measure the alignment between requirements concepts and architecture, in practice we found that requirements are not perfectly atomic. Therefore we introduce the term "needs" in Section 5 in order to define metrics for the alignment of the concepts. This allows generalizing the metrics to other requirements concepts, as it covers the number of needs detailed in a requirement resulting in notation independence. We further detail the link between needs in Section 5, and provide an example in Table 2.

Refinement. High-level requirements and architecture components are decomposed into detailed[4] requirements and architecture components, respectively [33].

Abstraction. Detailed requirements are grouped using high-level requirements, while detailed architecture components are bundled together based on similar functionality and placed in high-level architecture components [33].

[4] We use the term *detailed* instead of *low-level*, to avoid interpretations in which more refined requirements and components are conceived as being less valuable.

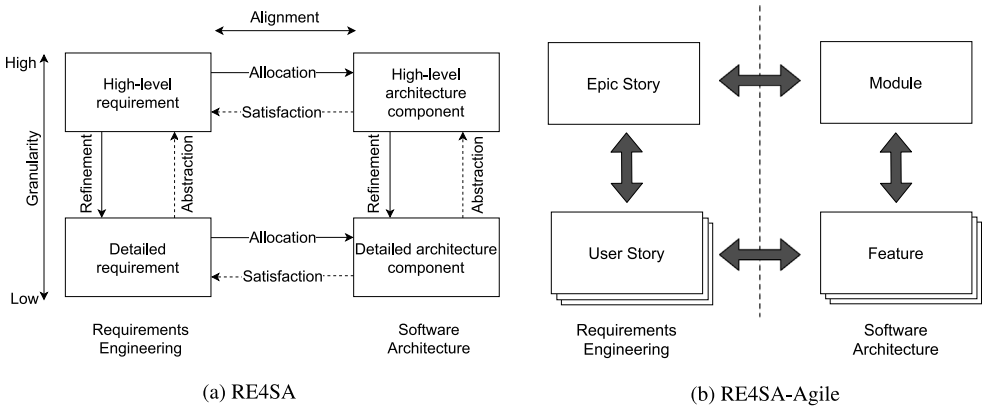


Fig. 2. The general RE4SA model and its instantiation for Agile development. In RE4SA, the solid arrows belong to the Architecture Discovery process, while the dashed arrows are part of the Architecture Recovery process.

Allocation. The process of relating requirements to architectural components is "*the assignment to architecture components responsible for satisfying the requirements*" [33]. Since both requirements and architectural components exist on two levels of granularity, this relationship is included on both levels.

Satisfaction. The SWEBOK guide states that "*the process of analyzing and elaborating the requirements demands that the architecture/design components that will be responsible for satisfying the requirements be identified*" [33]. Therefore, we refer to this relationship from architectural components to requirements as satisfaction.

Although there are ontological differences between the refinement and abstraction relationship depending on their application in the RE or SA domain, we use generic terms with low ontological commitment to avoid introducing too many, distinct terms in our RE4SA model.

3.1. Architecture discovery and architecture recovery

The RE4SA model, Fig. 2(a), supports the establishment of relationships between the four concepts in two ways: (i) Architecture Discovery (AD), a top-down process that takes the requirements as the input for creating an architecture; and (ii) Architecture Recovery (AR), a bottom-up process that first extracts the architecture from an implemented system [58], and then allows linking the architectural components to requirements.

The **AD process** (solid arrows in Fig. 2(a)) aims to design an intended architecture based on the requirements. It is advisable to start at the highest level of granularity, since the high-level requirements describe the functionality of the entire system, while on the lower level the details of this functionality is specified. Once the requirements have been defined, they can be allocated to architectural components. We suggest starting at the highest level: high-level requirements are allocated to high-level architectural components. Finally, it is useful to check if all the detailed architecture components included in the SA are represented in the detailed requirements. Detailed components that cannot be linked to a requirement may indicate missing requirements or unnecessary components.

The goal of an **AR process** (dashed arrows in Fig. 2(a)), instead, is to recover the implemented architecture from the system, using available documentation, such as source code and a run-time version of the system, and linking the recovered components to requirements. We suggest starting at the lowest level of granularity, and documenting the identified detailed architecture components. High-level architecture components can then be defined to group the detailed components.

AR is often an exploratory process that suggests a structured manner to analyze an existing software product. While we employed a simple process in previous publications [17,18], which starts from an analysis of the elements in the GUI of the product, we have since then identified the necessity for a more elaborate approach. In particular, the major challenge we found out is that the recovered detailed architecture components were too low level, and would not match the granularity of the requirements. We describe our revised process, used to achieve similar granularity levels, in Section 6.1.

The recovered architectural components can then be linked to requirements by creating satisfaction links. We recommend starting at the highest level of granularity: in RE4SA-Agile, the ES-module alignment. If these relationships are established first, it should be easier to identify which feature satisfies which US, for the USs are abstracted to ESs. Optionally, missing ESs or USs can be formulated, if the module or feature they will be allocated to is still relevant and/or required. On the other hand, ESs or USs that cannot be allocated to an architectural component need to be assessed. If the functionality the requirement describes is not required or desired, the requirement can be removed. If the opposite is true, the implementation of the feature(s) that would satisfy the requirement can be added to the backlog.

3.2. RE4SA-Agile

RE4SA-Agile is an instance of RE4SA that we constructed in collaboration with industrial partners, using concepts that we often found employed in agile practices [18,59]. On the requirements side, RE4SA-Agile uses one of the most common requirements representations in agile practices, User Stories (USs) [60,61]. In practice, USs are often grouped together using themes, epics or 'large USs'[62]. However, themes and epics tend to consist of one or a few words and thus lack the rationale that justifies why a requirement should be satisfied by the system [63]. Therefore, we propose the use of Epic Stories (ESs) [59], which make use of a clear template including both a motivation aspect and an expected outcome.

To illustrate the requirements side of RE4SA-Agile, let us consider a route planner application. A high level requirement using the ES template could be "*When I have to go to a place I don't know, I want to have a route planned for me, so that I can plan my trip and find the location.*" This ES can be refined in a set of USs, but for the sake of this example we will provide three such USs, which are shown in Fig. 3.

The RE4SA model assumes the existence of links between the requirements and the architecture concepts with similar granularity levels. From the architectural standpoint, we take the notion of 'module' from the functional architecture framework [39] as a grouping of features, which also allows for the visualization of usage scenarios through information flows [64]. A US describes a requirement for one feature [61]. Features are often represented using feature diagrams, a graphical language for organizing features hierarchically [65].

ES1	***When I have to go to a place I don't know, I want to have a route planned for me, so that I can plan my trip and find the location.***
US1	*As a consultant, I want to see all possible routes and select the fastest route to my destination, so that I can minimize my travel time when visiting customers*
US2	*As a car driver, I want to see the road and traffic information about the routes, so that I can prevent frustration*
US3	*As a car driver, I want to see the traffic jams on my route, so that I can prevent getting stuck in traffic*

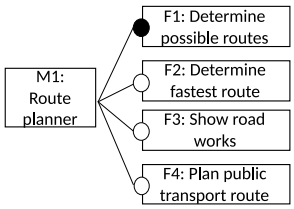


Fig. 3. Illustration of RE4SA-Agile for a route planner; on the left, one epic story is refined into three user stories; on the right, the feature diagram shows one module that is refined into four features.

For the ES presented in the example, we could design the module Route planner in our application that has the following features: Determine possible routes, Determine fastest route, Show road works, and Plan public transport route. Note that the example is purposefully not aligned with the requirements, so this can be discussed in the section on alignment metrics.

4. Granularity metrics

We analyze the *refinement* and *abstraction* relationships of RE4SA, which apply to both requirements and architecture. As explained in Section 3, granularity metrics determine the degree to which a high-level element is refined into detailed elements. In RE4SA, an element is either a requirement or an architectural component. Given a set of high-level elements $H = \{h_1, \ldots, h_n\}$ and a set of detailed elements $L = \{l_1, \ldots, l_m\}$, we can formally define refinement as a function $refines : H \rightarrow 2^L$, and $refines(h) = L'$, with $L' \subseteq L$.

Definition 1 (*Out-degree*)**.** Given a high-level element $h \in H$, the out-degree of h is the number of detailed elements in L that are a refinement of h. Formally, $out\text{-}degree(h) = |\{l.\ l \in refines(h)\}|$.

For example, in Fig. 3, we have that the out-degree of the epic story (the high-level requirement) is 3, for there are three user stories (detailed requirements) that refine it. On the right-hand side of the same figure, we have that the out-degree of the module *Route planner* (the high-level component) is 4, for there are four features (detailed components) that refine it.

The mean of the out-degrees then functions as an "ideal" granularity value that has been established as a convention by the requirements engineers and software architects. While expanding the functionality of a software product over various releases, deviations from this mean can trigger discussions to combine or split high-level components.

The out-degree of an individual high-level requirement or component is not a meaningful tool to measure granularity: for example, the fact that all high-level requirements are split into ten detailed requirements may be due to team conventions or company guidelines. We are interested in the identification of disproportionately large or small requirements or components with respect to the norm. These deviations do not determine an *error*, but rather a *warning*, a smell [66], that should be investigated by the product team.

We build on *outlier detection*, the set of statistical techniques that aim to identify elements that differ significantly from the majority of the data. We apply the Z-Score metric to our context as a simple metric that normalizes the data with respect to mean and standard deviation.

Definition 2 (*Granularity Score*)**.** Given H, L, and a high-level element $h \in H$, we define the granularity score G_h for the element h by applying the Z-score formula:

$$G_h = \frac{out\text{-}degree(h) - mean\{out\text{-}degree(h').\ h' \in H\}}{stddev\{out\text{-}degree(h').\ h' \in H\}}$$

A G-Score (similar as the Z-score) of 0 means that the granularity of an element corresponds to the arithmetic mean of the granularity in H. In statistics, outliers are identified when the G-score is above 3 or below -3. This is based on the so-called empirical rule, saying that in a normal distribution, approximately 99.7% of the measurements fall within three standard deviations from the mean.

However, our purpose is not that of excluding outliers from statistical analysis, but rather that of identifying high-level elements (requirements and components) that require attention and may be reworked, e.g., refactored. Furthermore, we cannot assume our data is normally distributed. Therefore, we take the G-Score as a basis but employ the following rules in order to identify *granularity smells*:

Definition 3 (*Granularity Smell*)**.** Given a high-level element h, take two real numbers λ and μ, with $\lambda < \mu$, which represent the light smell threshold and the severe smell threshold, respectively. We define four types of granularity smells:

1. severe under-granularity: if $out\text{-}degree(h) < 2$ or if $G_h \leq -\mu$;
2. light under-granularity: if $G_h \in (-\mu, -\lambda]$;
3. light over-granularity: if $G_h \in [\lambda, \mu)$;
4. severe over-granularity: if $G_h \geq \mu$.

Refinement to zero or one detailed elements means that the high-level element is not necessary, unless the high-level element is still incomplete. This indicates a severe under-granularity smell. This situation occurs also when the granularity of h is significantly smaller than the mean: $G_h \leq -\mu$. A light under-granularity smell happens when G_h is between the two real numbers $-\mu$ and $-\lambda$, indicating that the granularity of h is smaller than the mean. Conversely, if G_h is between λ and μ, then we obtain a light over-granularity smell. Finally, if G_h is higher than μ, we have a severe over-granularity smell, for the granularity of h is considerably higher than the mean. In the $(-\lambda, \lambda)$ interval, instead, we have cases of good refinement practices that do not lead to under- or over-granularity.

Fig. 4 illustrates the granularity smells for a set of requirements (excluding the case of $out\text{-}degree < 2$). The negative scores indicate under-granularity, the positive scores indicate over-granularity. To determine the granularity score bounds λ and μ, we have applied the granularity smells metrics to eleven datasets (see Table 1), all public except for DS11, in order to visually identify sensible values. These sets contain requirement artifacts, using different types of grouping. All detailed requirements were in the US format. The table briefly describes each dataset, its size, shows the number of smells that were detected using the G-score bounds, and mean and standard deviation for the out degree in a dataset.

As a result of our analysis, we set $\lambda = 1.1$ and $\mu = 1.7$, as they seemed adequate to pinpoint disproportionately large or small high-level requirements in the set. Our values of λ and μ constitute an initial baseline for future research. Also observe the high variation in the arithmetic mean and standard deviation, which confirm the suitability of an approach like ours based on the Z-score, rather than relying on an absolute number of detailed elements to denote smells.

Granularity resonates with the 'God element' phenomenon in software architecture [14], which occurs when an architectural element contains a disproportionately high amount of functionality than other elements: $G_{God\,Element} > \mu$. In our case, it happens when a high-level component is refined to significantly more detailed components than other high-level components of the system.

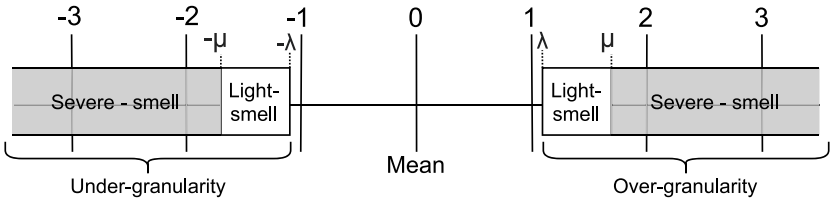


Fig. 4. Granularity score values with smell intervals. In our research $\lambda = 1.1$ and $\mu = 1.7$.

Table 1
Investigation of granularity smells in requirements data set; the results were used to define the smell thresholds λ and μ.

ID	Name	Description	Requirements		Smells		Out-degree	
			Low	High	Light	Severe	Mean	Std
DS01	Loudoun	Electronic land management system	60	11	1	1	5.45	3.33
DS02	ScrumAlliance1	First version of ScrumAlliance website	98	13	4	1	7.54	4.81
DS03	ScrumAlliance2	Video training website	73	15	2	1	4.87	3.36
DS04	Planning poker	Planning poker website	53	4	1	0	13.25	10.24
DS05	DataHub	Data sharing and management platform	65	8	3	0	8.13	4.94
DS06	MIS	University management information system	72	9	1	1	8.00	6.08
DS07	Cask	Big data integration platform	65	3	1	0	21.67	16.26
DS08	Duraspace	Opensource repository	104	25	1	2	4.16	4.50
DS09	RAC DAM	Software for data archiving and sharing	146	9	1	1	16.22	11.78
DS10	Zooniverse	Online citizen science platform						
	Counting #1	General USs as 'High'	33	8	0	1	4.13	4.19
	Counting #2	Include general USs in all groups	117	7	2	0	16.71	1.38
DS11	Remittance	Automatic remittance processing tool	51	7	1	0	7.29	3.09

5. Alignment metrics

We introduce metrics that allow for quantitative investigation of the relationship between requirements and architecture through the lenses of the RE4SA model. In particular, these metrics allow exploring the allocation and satisfaction relationships (see Section 3). As we introduce the metrics, they will be applied to the Route Planner illustrative example.

Let $R = \{r_1, r_2, \ldots, r_n\}$ be a collection of requirements and $C = \{c_1, \ldots, c_m\}$ be a collection of architectural components. In the RE4SA-Agile model, a requirement can be either an Epic Story (ES) or a User Story (US), while a component can be either a module or a feature.

Since a requirement can denote multiple needs in a part-whole fashion [67] (e.g., the conjunction 'and' is often used to express many needs within the same requirement [61,68]), we introduce the function $needs : R \to 2^C$ that maps a requirement r to the needs it expresses. Formally, given a set of needs N, we have that for any $r \in R$, $needs(r) = \{n \in N.\ requested\_by(n, r)\}$, where $requested\_by(n, r)$ is true when n is expressed in the text of requirement r. In this paper, the identification of the needs that are requested by a requirement is left to human analysis. For example, the user story "*As a consultant, I want to see all possible routes and select the fastest route to my destination, so that I can minimize my travel time when visiting customers*" can indicate two needs, Determine possible routes and Determine fastest route.

We can now introduce the set $N_R = \bigcup_{r \in R} needs(r)$ as the collection of needs that are requested by individual requirements in the set R. Similar to the $requested\_by$ predicate, we rely on human analysis for identifying the needs within a requirement, although linguistic techniques could be employed to locate the needs (e.g., the AQUSA tool can help locate non-atomic user stories [61]).

Definition 4 (*Alignment Matrix*)**.** An alignment matrix $A = (a_{ij})$ is a matrix of size $|N_R| \times |C|$ such that $a_{ij} = 1$ if and only if the need $n_i \in N_R$ *matches* the component $c_j \in C$. Formally,

$$a_{ij} = \begin{cases} 1, & \text{if } matches(n_i, c_j) \\ 0, & \text{otherwise.} \end{cases}$$

The alignment matrix can be used to explore the mutual relationship between requirements and components. Based on the matrix, we define $allocation : R \to 2^C$ as a function that returns the set of components that match the needs in a requirement (the *matches* predicate is also based on human mapping). Formally, $allocation(r) = \bigcup_{n_i \in needs(r)} \{c_j.\ a_{ij} = 1\}$. Conversely, we define a function $satisfaction : C \to 2^R$ that returns all the requirements with needs matching a given component: $satisfaction(c_j) = \bigcup_{r \in R} \{n_i.\ a_{ij} = 1 \wedge n_i \in needs(r)\}$.

The allocation function allows us to partition the set of requirements into four non-disjoint subsets: $R = R_{not} \cup R_{under} \cup R_{exact} \cup R_{multi}$, defined as follows:

- $R_{not} = \{r \in R.\ allocation(r) = \emptyset\}$
- $R_{under} = \{r \in R.\ 0 < |allocation(r)| \wedge \exists n_i \in needs(r).\ (\sum_j a_{ij}) = 0\}$
- $R_{exact} = \{r \in R.\ \forall n_i \in needs(r).\ (\sum_j a_{ij}) = 1\}$
- $R_{multi} = \{r \in R.\ \exists n_i \in needs(r).\ (\sum_j a_{ij}) > 1\}$.

R_{not} is the set of requirements that are not allocated, R_{under} are those requirements with some but not all allocated needs, R_{exact} are those requirements with each need allocated to exactly one component, and R_{multi} are those requirements having at least one need allocated to multiple components. The four sets are not disjoint. For example, a requirement requesting needs n_1 and n_2, with n_1 matching components c_1 and c_2 and with n_2 matching no components would be both multi-allocated (because of n_1) and under-allocated (because of n_2).

If we look at the example matrix in Table 2, US1 is exactly allocated, as it details two needs, each allocated to one feature. US2 is under-allocated, as it has two needs; one for traffic information, and one for road information, with only the road information need being met by the Show road works feature. US3 is not allocated, as no feature satisfies the need see the traffic jams.

Definition 5 (*Allocation Degrees*)**.** The partitioning of R into R_{not}, R_{under}, etc. can be used to define metrics on the allocation degree of a set of requirements. We introduce four degrees, each in the $[0, 1]$ range:

- multi-allocation degree: $multi\_alloc_d = |R_{multi}| \ / \ |R|$
- exact allocation degree: $exact\_alloc_d = |R_{exact}| \ / \ |R|$
- under-allocation degree: $under\_alloc_d = (|R_{not}| + |R_{under}|) \ / \ |R|$
- need allocation degree: $need\_all_d = \dfrac{|\{n_i \in N_R.\ (\sum_j a_{ij}) = 1\}|}{|N_R|}$

Table 2
Alignment matrix example from user stories to features.

User story	Need	Determine possible routes	Determine fastest route	Show road works	Plan public transport route	Allocation
US1: As a consultant, I want to **see all possible routes** and **select the fastest route** to my destination, so that I can minimize my travel time when visiting customers	N1	1	–	–	–	Exact
	N2	–	1	–	–	
US2: As a car driver, I want to **see the road** and **traffic information** about the routes, so that I can prevent frustration	N3	–	–	1	–	Under
	N4	–	–	–	–	
US3: As a car driver, I want to **see the traffic jams** on my route, so that I can prevent getting stuck in traffic	N5	–	–	–	–	Not

The ideal case is one in which the exact allocation degree and the need allocation degree are close to 1, and the multi/under allocation degrees are close to zero. In that case, indeed, each need in a requirement can be traced to almost exactly one architectural component. This situation is good because the needs are homomorphically mirrored in the architectural design, thereby facilitating the conversation between experts in either discipline. An exception to this case is when the system includes variability: in that case, it is desired to have a multi-allocation degree, for multiple components may be devised as alternative ways to fulfill one requirement. The need allocation degree is a need-level version of the exact allocation degree: it represents the ratio of needs that are allocated to exactly one component.

In the example of Table 2, we have a total of 3 requirements so the $|R|$ value is 3. US1 has exact allocation as both needs are met, US2 is under allocated as it has two needs of which one is met and US3 is not allocated as none of its needs are met. Therefore, our $exact\_alloc_d$ is $1/3 = 0.33$, our $under\_alloc_d$ is $(1+1)/3 = 0.67$, and $multi\_alloc_d = 0$. Regarding the need allocation degree: out of the five needs, three are met, which makes the $need\_all_d = 3/5 = 0.6$.

Similar to the partitioning of requirements based on the allocation function, we can partition the set of components based on the satisfaction function. Specifically, the set of components is partitioned into two disjoint subsets: $C = C_{not} \cup C_{sat}$, where $C_{sat} = \{c \in C.\ satisfaction(c) \neq \emptyset\}$ and $C_{not} = C \setminus C_{sat}$.

Definition 6 (*Satisfaction Degree*)**.** It defines the ratio of components that satisfy at least one need in a requirement as follows: $sat_d = |C_{sat}| \ / \ |C|$.

When the satisfaction degree reaches the value of 1, all architectural components trace back to at least one requirement and, thus, their existence is justified. Unlike Definition 5, we do not include a notion of multi-satisfaction, for we are interested in assessing *whether* a component is justified or not, instead of *counting* how many needs the component accommodates.

If we once again consider Table 2, three out of four features satisfy a requirement, the only feature that does not satisfy a requirement is the Plan public transport route feature. Therefore, the satisfaction degree is $3/4 = 0.75$.

We combine allocation and satisfaction into the metric of alignment, which is a weighted arithmetic mean of the extent to which needs are allocated, and the extent to which components can be traced back to requirements.

Definition 7 (*Alignment Degree*)**.** It is a weighted arithmetic mean ($\alpha \in [0, 1]$) of the need allocation degree and the component satisfaction degree: $align_d = \alpha \cdot need\_all_d + (1-\alpha) \cdot sat_d$.

In this paper, we set $\alpha = 0.5$ and give equal weight to the requirements and architecture perspectives. Similar to the debate on the β in the F_β-score regarding measuring the effectiveness of automated tools for RE [69], in-vivo studies are necessary to tune our parameter based on the relative impact of need allocation degree and component satisfaction degree. However, our experience with the software production industry reveals that early product releases include several implicitly expressed needs (e.g., printing, storage, menu interaction), thereby requiring a high $\alpha > 0.5$, whereas later releases focus on explicit (customer) requirements allocation with $\alpha < 0.5$.

To calculate the alignment degree of the example set in Table 2, given an $\alpha = 0.5$, we combine the need allocation degree (0.6) with the satisfaction degree (0.75). This results in the following: $0.5 \cdot 0.6 + (1 - 0.5) \cdot 0.75 = 0.675$.

The concepts and definitions above apply to the generic notions of *requirement* and *component*. In RE4SA, as per Fig. 2(a), we can reason about alignment at two granularity levels: *high* and *detailed*. The definitions and metrics can therefore be applied at either level:

- *high*: the set R contains ESs, C includes modules, N consists of outcomes from an ES, and the function *needs* returns the set of outcomes of an ES;
- *detailed*: R contains USs, C consists of features, N includes actions from a US, and the function *needs* returns the set of actions of a US.

6. The RE4SA-Agile model in practice

To assess the feasibility and usefulness of RE4SA and of our metrics, we applied them to two case studies. Both cases use the concepts as defined in the RE4SA-Agile model. The first case presents an AD process, while the second illustrates an AR process. After introducing each case, we present the case study approach in Section 6.1, analyze the granularity metrics in Section 6.2, and then report on the alignment metrics in Section 6.3.

Vendor Portal (VP). The discovery case concerns a portal for vendors to manage their open invoices through an integration with the customers' ERP system. The dataset contained 30 user stories, and while the project was ongoing at the time of writing, the main functionalities, 35 features contained in 13 modules, where delivered in four weeks with five active team members. The development was done using a low-code platform. Following a requirements elicitation session with the customer, a list of USs was created and then grouped in themes. We defined 8 ESs from the themes by rewording them and by splitting one of them into two (based on the word "and"). The software architecture was created by transforming the requirements into an intended architecture following the AD process described in Section 3.1. The software architect was allowed to include his interpretation of the requirements, e.g., by adding missing features and modules.

Fig. 5 shows how USs were allocated to features. The US1 in the figure is multi-allocated, as it is linked to two features, specifically the need "*use password forgotten functionality*" is allocated to the features "*initiate password recovery*", and "*send password recovery email*". The other two USs are exact-allocated as they contain a single need and are allocated to a single feature.

Your Data (YODA). The recovery case regards a research workspace developed for Utrecht University.[5] A collection of 58 USs was provided

[5] https://github.com/UtrechtUniversity/yoda/

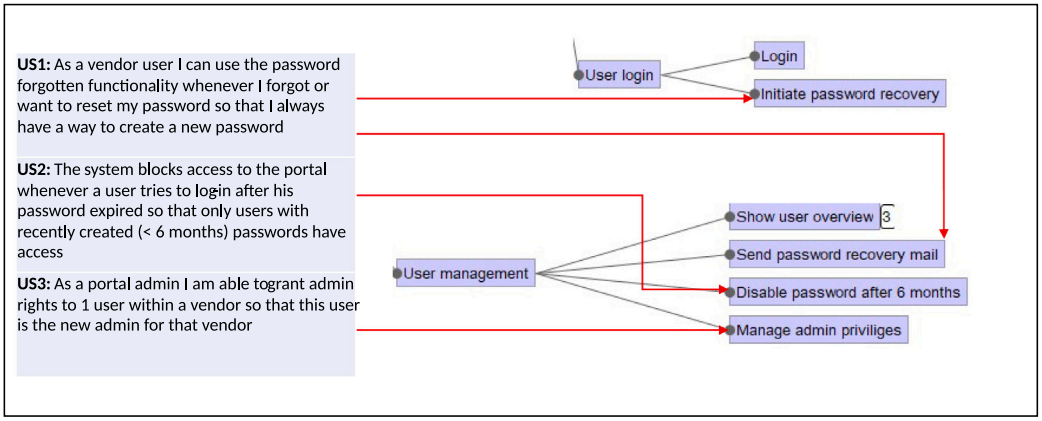


Fig. 5. Example of how USs were allocated to features for the VP case.

to us by the development team, already grouped in themes. We used these one-word themes to formulate 10 ESs. The functional architecture had to be recovered [which resulted in 60 features contained in 10 modules]. As described in Section 3.1, this was done using a bottom-up approach. Using the implemented system, in this particular case a web application, all features were recovered by modeling every user-interactive element in the GUI as a feature.

An example of how modules and features were recovered from the GUI is shown in Fig. 6. For the sake of brevity, the alternative features related to F2 and F3 were collapsed. The module satisfies an ES that was based on the Metadata theme: "*When I am storing research data, I want to include metadata about the content, so that I can document my data.*" Only two of the features satisfy a US, features F3 and F4 (in Fig. 6) satisfy US3 and US4, respectively:

US3: "*As a researcher, I want to specify the accessibility of the metadata of my dataset, so that access can be granted according to policy [...].*"

US4: "*As a researcher, I want to be able to discard existing metadata and re-begin adding metadata, so that I can document a data package.*"

Therefore, considering our metrics for determining the satisfaction degree (Definition 6), F1 and F2 are part of the C_{not} count, while F3 and F4 are part of C_{sat}.

6.1. Case study approach

In this section, we discuss a secondary investigation of the case studies: we performed further investigation and exploration of the data sets discussed in previous research [17]. This was mostly inspired by the addition of the granularity metrics, and the changes from this new perspective.

Following the guidelines by Wohlin et al. [70], we discuss the approach and goals for the case studies. We report on those case study findings that are interesting and relevant for this research, and have to omit some details due to confidential agreements with the data sources.

The *object of study* is the metrics defined in Sections 4 and 5. Our *purpose* is that of evaluating the use and effectiveness of the metrics in industry scenarios. The cases start help us explore specific aspects of the research questions RQ1 and RQ2 that we listed in the introduction. In particular, we hypothesize that our metrics are an effective tool to analyze granularity and alignment. We set the following hypotheses H1 and H2, which relate to RQ1 and RQ2, respectively.

H1. Granularity smells pinpoint opportunities for achieving uniform granularity within and between functional requirements and architecture specification.

H2. Allocation, satisfaction, and alignment degrees pinpoint opportunities for establishing and maintaining alignment between functional requirements and architecture specification.

The *context selection* was done based on convenience selection: for the VP case, one of the researchers was embedded in the organization; for the YODA case, we had a connection to the development team. While the cases are both selected from convenience sampling, they are different in nature; VP is a commercial product, while YODA is an academic project. For further triangulation in our research, the cases have been investigated by a principal investigator and validated by a secondary investigator.

Data collection was a combination of second degree (indirect involvement of the researchers) and third degree (study of the work artifacts only) techniques [71]. We collected architectural data for YODA by researching the software product via a second degree collection method, through the architecture recovery from the UI and documentation. For the requirements and for the VP architecture, we employed a third degree collection technique, as we analyzed completed artifacts: requirements specification for both cases and feature diagram for the VP case.

To limit validity threats, we employed multiple techniques. To support replication, we provide the used data in our online appendix. We complement the second and third degree data collection techniques, which did not involve interaction with the stakeholders, with sessions in which we discussed the findings with the stakeholders of the investigated artifacts. Additionally, we reflect on the findings that result from the application of the metrics. Despite our attempts, some threats could not be prevented, partially due to the convenience selection of the case study materials. Considering the exploratory nature of the case studies, the results are not conclusive but show an initial application of the introduced metrics. The validity threats are further detailed in Section 7.

6.1.1. Changes from previous investigation

While exploring the cases regarding the granularity metrics of Section 4, we determined that there was a discrepancy with what the feature diagrams contained as feature, and what the researchers considered a feature. It was decided that the cases would be revised, compared to our previous work [17], to achieve a more standard feature diagram granularity:

- **Differentiating features**: any feature is included that represents a key functionality of the software product that offers competitive advantage and helps differentiation from the market competitors [72]. The granularity of a differentiating feature depends on the product domain.
- **Information hiding**: the features that are part of an "alternative" or "OR" decomposition, which are fundamental to represent variability, are counted as one composite feature for our purposes. This allows for separating the internal and external structure and/or behavior [73]. To illustrate, in case of language selection,

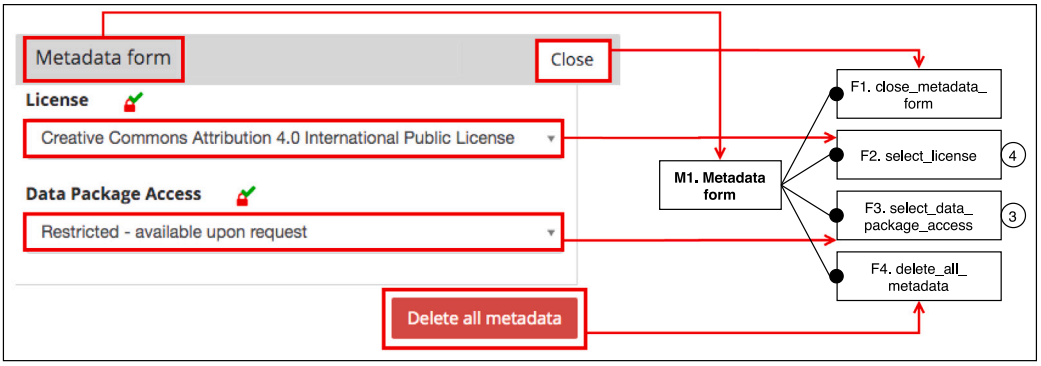


Fig. 6. Example of architectural component recovery from the GUI for the YODA case.

it is of importance to know that language can be selected, not which languages are included. The reason for this exclusion is that one of the alternative features is picked on deployment, so the additional features do not increase the system or functionality from a user perspective.

- **Value**: breaking down a feature into parts leads to individual features that in isolation do not have value for the system. For example the feature display vendor address has more value for the VP case, while the individual features display vendor zip code and display vendor address line in isolation do not. In these cases, the composite feature is the lowest level of depth included in the feature diagram. This is based on the communication unity [73].
- **Role-based features**: Two similar features may have different use cases as users with different roles use the features. For example a system admin might be able to manage password for all users in the system and set up a new one if a user contacts them. While the manage password in a profile module will only allow a user to change their own password. These features will be counted separately.

Applying these guidelines to the case studies has changed the number of features, which in turn impacts the alignment metrics. Since there is still a subjectivity in the metric calculations, we have opted to make the case study data sets public. This way other researchers can refer to our data and compare it to their own calculations. This data set can be found in a footnote in the introduction.

6.2. Granularity: Studying refinement and abstraction

We apply the granularity metrics as defined in Section 4 to the VP and YODA cases. Granularity scores (G-score) between the **1.1** and **1.7** thresholds are expressed in bold and indicate a light smell, while those above the **1.7** threshold, or containing a single element are expressed in bold with a gray background and indicate a severe smell.

VP. Table 3 reports the granularity metric scores applied to the VP case. The set contains two severe over-granularity smells (ES4 & M7), one in the requirements and one in the modules. The two severe smells are related, as ES4 is uniquely allocated to M7. This indicates that detection and reaction to the smell on the requirements side would allow prevention of the architecture smell. The three light smells in granularity score for ES6, M5 and M6 are for high-level components that only contain a single detailed component. However, as discussed in Section 4, a single component is also a severe smell. This can indicate an inconsistent granularity level, or missing detailed components. Specifically, the user story in ES6 was only linked to a single feature in the alignment matrix, indicating that the Epic (or theme it was based on) should have just been a single user story. M5 represents the module asset management and M6 represents invoice status overview. M5 was a specific module that allowed users to download files, an optional element in the use cases. In future sprints, this module might be extended, e.g., with file previews, or sharing possibilities. From the point of view of the researchers, M6 could be combined with M3 Vendor overview as a feature showing the invoices per status for a single vendor. Thus it might be on the wrong level of granularity, unless the product evolution is expected to add features to this module.

YODA. The granularity metrics for the YODA case are presented in Table 4.

On the requirements side, two ESs have an disproportionate out-degree when taking all ESs and USs into account. ES1 has a light granularity smell, with a granularity score of **1.61** and ES10 has a severe smell, with a score of **1.84**.

Especially the tenth module, M10 in Table 4 is cause for concern. Its out-degree is four times as high the second largest module. The high number of features may indicate that this module contains many functionalities and therefore has too many responsibilities, which is generally ill-advised. There is a risk that this module is or may become a God-element, potentially leading to a bottleneck in the system [14]. At this point, the module can be considered saturated, so no features should be added to it. If functionality does need to be added, the module should be split first.

In upcoming updates or releases of the system, this module may need refactoring to decrease the out-degree. For example, by assigning responsibilities to other modules or by splitting it into two modules, to prevent it from becoming too large. This risk could have been prevented or mitigated, as the requirement granularity scores already indicated that ES (and later module) 10 was relatively large.

6.3. Alignment: Studying allocation and satisfaction

The alignment metrics for both cases are presented in Table 5, including both the ES-module alignment and the US-feature alignment. The allocation and satisfaction for both cases have been performed by two of the researchers, and any discrepancies in tagging have been discussed and resolved. One point of discussion was the inclusion of needs stated in the "so that..". part of USs, as these are only meant to indicate motivation. However, we decided to include all needs specified, as requirement sets are generally not perfect, and these still indicate needs in the documentation.

VP. In the VP case, the ES-module alignment score is 0.92; while all needs are allocated, two of the modules do not satisfy a requirement. Additionally, three of the ESs (ES 2, 7, 8 in Table 3) are multi-allocated, with ES2 being allocated to 5 modules (M2-M5) this multi-allocation indicates that the ES encapsulates too many of the functionalities in the set. Although the case company had no unity criteria for the requirements or architecture, these findings indicate that the functional requirements and architecture artifacts did not have a consistent level of granularity. This can be used in further improvements of the artifacts; to ease the maintenance of RE and SA artifacts as they co-evolve, the requirements can be split to match the level of granularity of the

Table 3
The granularity-related metrics applied to the VP case.

Epic Story	ES1	ES2	ES3	ES4	ES5	ES6	ES7	ES8						Mean	Std
# USs	5	6	4	8	2	**1**	2	2						3.75	2.43
G-score	0.51	0.92	0.10	**1.75**	−0.72	**−1.13**	−0.72	−0.72						–	–
Module	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	Mean	Std
# Features	2	4	5	2	**1**	**1**	6	2	2	3	3	2	2	2.69	1.49
G-score	−0.46	0.87	**1.55**	−0.46	**−1.13**	**−1.13**	**2.21**	−0.46	−0.46	0.21	0.21	−0.46	−0.46	–	–

Table 4
The granularity-related metrics applied to the YODA case.

Epic Story	ES1	ES2	ES3	ES4	ES5	ES6	ES7	ES8	ES9	ES10	Mean	Std
# USs	13	3	6	7	2	6	2	3	2	14	5.8	4.47
G-score	**1.61**	−0.63	0.04	0.27	−0.85	0.04	−0.85	−0.63	−0.85	**1.84**	–	–
Module	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	Mean	Std
# Features	5	2	3	7	2	4	4	3	2	28	6	7.89
G-score	−0.13	−0.51	−0.38	0.13	−0.51	−0.25	−0.25	−0.38	−0.51	**2.79**	–	–

Table 5
The alignment-related metrics applied to the VP and YODA cases.

Relationship	Metric	VP		YODA	
		ES-M	*US-F*	*ES-M*	*US-F*
Allocation	$multi\_alloc_d$	0.38	0.17	0.0	0.10
	$exact\_alloc_d$	0.63	0.73	1.0	0.81
	$under\_alloc_d$	0.00	0.10	0.0	0.09
Satisfaction	sat_d	0.85	0.86	1.0	0.89
Alignment	$need\_all_d$	1.00	0.93	1.0	0.94
	$\boldsymbol{align_d}$	**0.92**	**0.89**	**1.0**	**0.91**

architecture. If we exclude ES2, the satisfaction drops from 0.85 to 0.69. This shows that a difference in the level of granularity between the artifacts impacts the scores.

On the other hand, one of the requirements (ES6) was too fine grained. This ES details an address change request, which can be traced to a single feature Request vendor information change. This was still seen as allocation, because although the module did more, it still fulfilled the need in the requirement. The satisfaction score indicates that around 15% of both granularity architecture components do not directly satisfy a requirement. The remaining components are not explicitly justified by the requirements.

Since this is an AD process, we expect a high alignment degree, as the architecture is based on the requirements before taking implementation factors into account (as opposed to the AR process). The alignment degree is around 0.9 on both granularity levels, indicating slight discrepancies between the requirements and the architecture. Together with the multi-allocation degrees of 0.38 and 0.17, this seems to indicate that *the requirements set is not sufficiently detailed*. Based on these results, communication between the requirements engineer and the architect can increase the consistency between the artifacts and lead to new requirements or architecture components. The inexact allocation on the ES-module level can indicate an incorrect categorization of requirements, that the granularity of ES is not on a module level, or that the architect's categorization differs from that of the requirements engineer.

The metrics from the VP case were discussed with the product owner of the portal, who was *surprised by the low alignment score*. Indeed, the project was rather simple and the requirements were the basis for the architecture. The architecture design already had an alignment score of 0.1 lower than a perfect alignment. The product owner indicated that the metrics can be used to identify potential issues with the requirements. Applying the metrics also determined that 2 requirements were not yet satisfied, which was then resolved in the architecture design. It was also noted that the requirements specification was not revisited after the SA creation, and based on the alignment degree, this might be an action point for the development team.

Multi-allocation was seen by the product owner as the most important allocation degree, as it *can indicate unnecessary costs*. Under-allocation was expected to be detected during use of the application, or denote missing features to add later. The modules that did not satisfy a requirement were judged to be a *result of missing requirements*. Finally, it was mentioned that the metrics can be used to make *agreements when outsourcing development*, e.g., requiring the architecture to have a 0.9 alignment degree with the requirements.

YODA. The ESs were allocated to exactly one module and the modules had a one-to-one relationship with ESs in terms of satisfaction. Therefore, the need allocation, satisfaction and alignment scores are all 1.0 and are not discussed any further. As evidenced by the satisfaction score in Table 5, almost every feature satisfied at least one need. Only seven features, out of 61, did not satisfy any needs. Furthermore, nearly all USs were allocated to a feature and only four needs were unallocated out of a total 65 needs. One US was under-allocated, meaning that some of its needs were allocated, but not all.

Five USs are severely multi-allocated: three USs can be allocated to nine features, while two others can be allocated to 28 and 26 features (these can be found in the YODA dataset alignment matrix with ID 13 & 18). When these USs are excluded from the alignment metrics, the results are clearly different. In both situations, there are 61 features. In the dataset as-is, 54 of those features satisfy a need, which leads to a satisfaction score of 0.89 and an overall alignment score of 0.91, as presented in Table 5. When the five aforementioned USs are removed from the alignment matrix, only 32 out of 61 features satisfy a need (satisfaction score = 0.52), which means that only half of the features satisfy at least one need. The need allocation score is only decreased by 0.01 (to 0.93) when using two decimals, which results in an alignment score of 0.73.

In the YODA case, we proposed a modularization, which was then attempted by the development team. However, due to the many technical dependencies in the technical architecture they could not apply the full modularization. This led them to refactor their software. This calls for research on the link between functional architecture and technical architecture from an alignment point of view. For instance, determining how the functional architecture is impacted by decisions in the technical architecture and whether a technical architecture can be designed based on the functional architecture.

Based on these findings, we hypothesize that the satisfaction score, and therefore the alignment score, can be misleading if severely multi-allocated requirements are included in the alignment matrix. We recommend *calculating the scores including and excluding severely multi-allocated requirements to measure their effect on the satisfaction and alignment scores*. In addition, we suspect that severely multi-allocated USs, such as the ones that could be linked to more than 20 features, are formulated using a level of granularity that is *dissimilar to the level of granularity used for the architectural components*. Arguably, the five USs that were previously mentioned should either be split into multiple USs or formulated as ESs. This indicates that they were underspecified compared to the rest of the requirements, one of the most common problems in RE practice [8].

The metrics reveal that *not all requirements are currently allocated*: some features still need to be implemented. Moreover, if the severely multi-allocated USs are excluded, nearly half of the features do not satisfy a need. So, either the requirements are incomplete or unnecessary features exist. The lead developer explained that they *do not consider anything in retrospect*: when a US is considered completed, it is removed from the backlog. Thus, he was *unaware that five USs have not yet been (fully) implemented in the system.*

According to YODA's lead developer, the metrics could prove to be useful in several ways. First, they could help *foster the creation of trace links*, currently nonexistent. When new colleagues join the team, it takes them "approximately three months to get up to speed and be able to add something of value to the system". Second, when someone leaves the team, their knowledge is lost. Also, team members often *do not know where features originate from*. Oftentimes, the rationale is unknown and the source code is checked to locate features; if unused, it is removed. The reason for this lack of documentation is that the team sometimes adds features without defining the requirements first. As reflected by the satisfaction scores, the team saw that they added features without documenting the requirements for them. Moreover, he expects the metrics to be of use for sprint reviews. Under-allocation, for instance, to *check whether all requirements were satisfied and if they were satisfied in full*. Finally, in an attempt to prevent the team from implementing the same feature twice, the multi-allocation metric can help them identify overlap in USs or even duplicate features. The developer stated they plan on *using the metrics in their next sprint aiming to improve their work efficiency and quality*.

7. Findings & validity

Applying the RE4SA-Agile model to two case studies has shown the importance of considering granularity in all stages of alignment evaluation. A mismatch in the granularity between the RE and SA sides can skew the alignment metrics. Both in the YODA case for US-feature and in the VP case for ES-module, we saw how a too coarse-grained requirement can lead to a satisfaction score that is much higher as components allocate to parts of the requirement. This can be prevented by having clear agreements on the unity criteria [73] used in the requirements and architecture on a project or product level. Furthermore, mismatches in granularity are sometimes indicated by the alignment metrics: a high multi-allocation degree can indicate that features are more fine-grained than the requirements. For instance, an incorrect categorization of high-level requirements can result in an inexact allocation of the high-level requirements and components. For example, in the YODA case, removing two coarse granulated user stories reduces the satisfaction degree from 0.89 to 0.75, and therefore the alignment degree from 0.91 to 0.85. This shows that severely multi-allocated requirements can heavily impact the satisfaction score and, therefore, the alignment score.

Finding 1

A difference in the level of granularity between artifacts can impact both the granularity and alignment scores. As a possible solution, we recommend calculating the metrics including and excluding severely multi-allocated requirements, to measure their impact.

When we consider H1, regarding the effectiveness of the granularity smells, we see that smells in the requirements were often accompanied by smells related to the corresponding architectural components. In the VP case, the severe smell in ES4 can be traced to the smell in M7. And in the YODA case, the severe smell in ES10 can be traced to the smell in M10. While these results are seen as a first study, and we believe additional case studies are required for confirmation, we summarize this in Finding 2:

Finding 2

The granularity scores of the requirements artifacts can indicate potential smells in the corresponding architecture specification. The development team can use the requirements granularity smells to predict architecture smells and be warned of them.

In our research, we somewhat simplified the connection between the concepts. For example, there are cases in which modules possess sub-modules (and even sub-sub-modules) or features are hierarchically organized. For our empirical investigation of the RE4SA model, we explicitly chose to work with a view of one level of decomposition in order to keep the metrics and links understandable. When applying the model to new cases, we suggest using unity criteria for determining which functional elements to use in the metrics calculations. For example, in the VP case, M1–M6 were sub-modules of the Back office module, which contained all back-end functionalities. However, if we had considered this a single module for the purposes of our metrics, it would have provided a skewed view on alignment and granularity, as these sub-modules are on a similar level of granularity as the other modules.

When we evaluate H2, regarding the alignment degree, we observe how the detection of a lower-than-expected alignment degree allows a project team to determine that they need to ensure that they have a shared understanding of the design for the application. In these scenarios, the metrics can be used to facilitate team communication by pinpointing misalignment. For example, the alignment score of 0.9 in the VP case before development even started, indicates that there are conflicting views on the system between the requirements engineer and software architect. The team can then collaborate to improve both the requirements and architecture of the solution, with the explicit goal of maximizing the alignment degree. Additionally, the conceptual link between specific concepts in the RE and SA domains create a common ground for the requirements engineer and software architect.

Finding 3

A low alignment score indicates a need for communication within the project team. Identifying misalignment can lead to activities to ensure the members of a project team are on the same page, mitigating one of the most prevalent causes of project failure [8].

According to practitioners, the alignment metrics (providing partial answer to H2) can be of use in several ways: to identify unnecessary costs, to identify missing requirements, to make agreements when outsourcing development, to support traceability and to check whether

all requirements were implemented. While using the metrics for outsourcing agreements would help ensure the specified functionalities are met, quality/non-functional aspects should be considered as well, as those define *how well* a functionality is implemented. This topic, which also requires considering the inter-dependencies among components, is left to future work. As indicated in the evaluation of the AR case, the alignment metrics could be integrated in development sprints to evaluate if the requirements of a sprint are met. This leads to an iterative use of the metrics on smaller sets of requirements, potentially increasing their usability for agile projects. Incorporating the metrics in a sprint can also facilitate the detection of trace links, as the allocation matrix indicates which components satisfy specific requirements.

Validity threats. In relation to *conclusion validity* we identify three threats. First, the results obtained from the analysis using the metrics are affected by the level of granularity that was selected. While we endeavored to adhere to the granularity levels used in the source material, different levels of granularity used may lead to different results. Second, the granularity scores are dependent on the mean and standard deviations of the dataset in question. If the requirements and/or architecture are revised/refactored according to the results, for instance a high-level architecture component is split to reduce its number of components, the metrics need to be recalculated. Due to the refactoring of, in this case, the module, the mean and standard deviation have been affected. Third, the metrics were applied to only two cases; although these are representative examples of software products, and several findings are shared, our findings mostly apply to those cases.

Concerning *internal validity*, similar to the previously mentioned conclusion validity threat, the selected level of granularity may affect the internal validity of the research as well. USs should describe a requirement for exactly one atomic feature, but this is not always the case due to inefficiency, meaning USs may describe composite features instead. The US "*As a user, I want to select a language*" would, theoretically, result in one feature select language. However, one may decide to link this US to all language options available. To mitigate this threat, we have formulated guidelines for feature diagram granularity standardization. In addition, the granularity and alignment metrics were cross-validated and the data-sets have been made available.

In terms of *construct validity*, the use of epic stories in the RE4SA-Agile model and case studies may bring some risk. In RE practice, ESs as presented here are rarely. ESs, or rather 'epics', are more often written using the US template or as themes (one or a few words). The re-formulation of epics and themes into ESs did not pose any particular challenges here, but it is possible that others would have formulated these ESs in a different manner. The other concepts included in the RE4SA-Agile model have already been adopted by practitioners. Furthermore, the abstraction relationship was only partially validated using the case studies. Both case studies included some type of grouping of requirements beforehand. Ideally, we would also investigate grouping sets of detailed requirements in high-level requirements from scratch. During the analysis of the datasets, there were a couple of discrepancies in the alignment tagging. These can be explained by the fact that one of the researchers was more familiar with the specific case. Additionally, for the VP case we had more in depth knowledge, as one of the authors is embedded in the case context. For the YODA case, we were limited to a number of conversations with the development team and our interpretation of the artifacts.

Finally, with regards to *external validity*, the testing of the metrics was limited to two cases. We did, however, apply the metrics to real-world documentation and base the granularity score bounds on 12 different data sets. In addition, the metrics and guidelines presented in this paper are meant for the assessment of requirements and architecture only. It is entirely possible that a product or system is non-problematic or without smells according to our metrics and guidelines, but not according to others. Also, while the alignment metrics were discussed with stakeholders related to the cases, the granularity metrics were not.

8. Conclusion

In this study on requirements and architecture alignment, we proposed the RE4SA model that provides a connection between artifacts and that facilitates communication within the development team. We formalized the links between the artifacts within the RE and SA domains at a conceptual level, and applied these notions to a specific instance of the model: the RE4SA-Agile model.

Additionally, we provided metrics to quantify the alignment between RE and SA and detect smells in granularity by focusing on outliers in a dataset. These metrics have been applied in two industry-provided cases and allow for detection of smells and for making improvements in both architecture and requirements. The metrics are also useful for detecting the need for communication within a project team during projects and in requirements and architecture reviews or revisions. Performing the explicit anomaly analysis for granularity and alignment of a software system assist RE and SA in establishing and maintaining well structured, traceable artifacts.

To answer RQ1 on how to assist in achieving uniform granularity, our proposed solution consists of applying the granularity metrics, to detect granularity smells as presented in Section 4. These metrics are expected to reveal opportunities for achieving uniform granularity (H1 in Section 6). While the smells do indicate such opportunities (e.g., we could identify God-element occurrences in the architecture of the YODA case), future work is necessary to assess whether re-establishing uniform granularity actually leads to better software systems. Moreover, in Section 7, we observed that it was often the case that granularity smells co-occurred in both the RE and the SA side together, thereby showing that analyzing granularity at the requirements level may prevent architectural granularity smells.

We address RQ2 on how to assist establishing and maintaining alignment through the allocation, satisfaction, and alignment metrics proposed in Section 5 (leading to H2 in Section 6). The metrics allowed to identify situations in which requirements were not allocated or modules were not justified; according to our interviewees, these issues may lead to unnecessary costs or to foster the creation of trace links. Finding 3 in Section 7 highlights how low alignment can be an indicator of need for better communication within the project team.

While this research has shown promising results, our results are still preliminary, and the findings need to be investigated more thoroughly to allow for generalization. For example, while both cases indicate that a granularity smell in the requirements leads to a similar granularity smell in the architecture, this needs to be researched empirically. Additionally, more extensive guidelines could be identified for the alignment metrics. For example, the notion of requirement multi-allocation could be formalized as it could cause issues related to under-specification [8].

Furthermore, the generalized RE4SA model has only been tested through the RE4SA-Agile instance. We invite other researchers to apply the model and the metrics with alternative RE and SA artifacts. Additionally, while we studied alignment between requirements and functional architecture, we surmise that this alignment may also be studied with respect to tests or code.

As stated, our research focused on the functional requirements and architecture. However, this is not the only perspective that can be considered for the alignment and granularity. Similar connections might be present between non-functional requirements and different architecture concepts. We do however, hypothesize that these architectural decisions are often made in early stages of design, and are less likely to change compared to functional concepts. For instance, the choice for a cloud platform like Azure, AWS or a low-code development platform like in the VP case, constrains further choices. Therefore, we expect that an initial architecture design can be mapped to the non-functional requirements to ensure the requirements are met.

The activities in this research were mostly performed manually. For future scenarios, we envision that software tools could assist the use of

the RE4SA model. For example, by relying on the linguistic structure of the artifacts, we could identify allocation and satisfaction links between the requirements and architecture.

Evolution of software products in agile environments [3] is a challenge that could benefit from application of RE4SA-Agile. By applying the metrics on a sprint basis, as suggested in the YODA case, the effort required is limited to the sprint scope. Additionally, this would ensure that the evolution of the software product becomes visible and manageable. Which in turn keeps the SA and RE documentation up to date.

CRediT authorship contribution statement

Tjerk Spijkman: Conceptualization, Methodology, Formal analysis, Investigation, Data curation, Writing, Visualization. **Sabine Molenaar:** Conceptualization, Methodology, Formal analysis, Investigation, Data curation, Writing, Visualization. **Fabiano Dalpiaz:** Conceptualization, Methodology, Formal analysis, Investigation, Writing, Visualization. **Sjaak Brinkkemper:** Conceptualization, Methodology, Formal analysis, Investigation, Writing, Visualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank fizor. and the YODA development team for their contributions in the evaluations and data for our case studies. Additionally, we would like to thank Sergio España for providing input for this research.

References

[1] B. Nuseibeh, Weaving together requirements and architectures, Computer 34 (3) (2001) 115–119.
[2] M. Galster, M. Mirakhorli, A. Koziolek, Twin Peaks goes agile, SIGSOFT Softw. Eng. Notes 40 (5) (2015) 47–49.
[3] J. Cleland-Huang, R.S. Hanmer, S. Supakkul, M. Mirakhorli, The Twin Peaks of requirements and architecture, IEEE Softw. 30 (2) (2013) 24–29.
[4] G. Lucassen, F. Dalpiaz, J.M. Van Der Werf, S. Brinkkemper, Bridging the Twin Peaks: The case of the software industry, in: Proc. of the International Workshop on the Twin Peaks of Requirements and Architecture, 2015, pp. 24–28.
[5] M. Galster, M. Mirakhorli, J. Cleland-Huang, J.E. Burge, X. Franch, R. Roshandel, P. Avgeriou, Views on software engineering from the Twin Peaks of requirements and architecture, ACM SIGSOFT Softw. Eng. Notes 38 (5) (2013) 40–42.
[6] J. Whitehead, Collaboration in software engineering: A roadmap, in: Proceedings of Future of Software Engineering, 2007, pp. 214–225.
[7] I.R. McChesney, S. Gallagher, Communication and co-ordination practices in software engineering projects, Inf. Softw. Technol. 46 (7) (2004) 473–489.
[8] D.M. Fernández, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetrò, T. Conte, M.-T. Christiansson, D. Greer, C. Lassenius, et al., Naming the pain in requirements engineering, Empir. Softw. Eng. 22 (5) (2017) 2298–2338.
[9] B. Curtis, H. Krasner, N. Iscoe, A field study of the software design process for large systems, Commun. ACM 31 (11) (1988) 1268–1287.
[10] D. Zowghi, N. Nurmuliani, A study of the impact of requirements volatility on software project performance, in: Proceedings of the Asia-Pacific Software Engineering Conference, 2002, pp. 3–11.
[11] C.C. Venters, R. Capilla, S. Betz, B. Penzenstadler, T. Crick, S. Crouch, E.Y. Nakagawa, C. Becker, C. Carrillo, Software sustainability: Research and practice from a software architecture viewpoint, J. Syst. Softw. 138 (2018) 174–188.
[12] M. Lindvall, D. Muthig, Bridging the software architecture gap, Computer 41 (6) (2008) 98–101.
[13] W. Bekkers, I. van de Weerd, M. Spruit, S. Brinkkemper, A framework for process improvement in software product management, in: Proceedings of the European Conference on Software Process Improvement, 2010, 1–12.
[14] N. Rozanski, E. Woods, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, Addison-Wesley, 2011.
[15] A. Moreira, J. Araújo, I. Brito, Crosscutting quality attributes for requirements engineering, in: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, 2002, pp. 167–174.
[16] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison-Wesley Professional, 2003.
[17] S. Molenaar, T. Spijkman, F. Dalpiaz, S. Brinkkemper, Explicit alignment of requirements and architecture in agile development, in: Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer, 2020, pp. 169–185.
[18] T. Spijkman, S. Brinkkemper, F. Dalpiaz, A.-F. Hemmer, R. van de Bospoort, Specification of requirements and software architecture for the customisation of enterprise software: A multi-case study based on the RE4SA model, in: 2019 IEEE 27th International Requirements Engineering Conference Workshops, REW, IEEE, 2019, pp. 64–73.
[19] T. Dingsøyr, D. Falessi, K. Power, Agile development at scale: The next frontier, IEEE Softw. 36 (2) (2019) 30–38.
[20] L. Cao, B. Ramesh, Agile requirements engineering practices: An empirical study, IEEE Softw. 25 (1) (2008) 60–67.
[21] J.O. Coplien, G. Bjørnvig, Lean Architecture, John Wiley & Sons, 2011.
[22] A. Rashid, A. Moreira, J. Araújo, Modularisation and composition of aspectual requirements, in: Proceedings of the International Conference on Aspect-Oriented Software Development, 2003, pp. 11–20.
[23] D. Ameller, C. Ayala, J. Cabot, X. Franch, Non-functional requirements in architectural decision making, IEEE Softw. 30 (2) (2012) 61–67.
[24] M.W. Whalen, A. Gacek, D. Cofer, A. Murugesan, M.P. Heimdahl, S. Rayadurgam, Your "what" is my "how": Iteration and hierarchy in system design, IEEE Softw. 30 (2) (2012) 54–60.
[25] K. Pohl, E. Sikora, COSMOD-RE: Supporting the co-design of requirements and architectural artifacts, in: Proceedings of the IEEE International Requirements Engineering Conference, 2007, pp. 258–261.
[26] P. Grünbacher, A. Egyed, N. Medvidovic, Reconciling software requirements and architectures with intermediate models, Softw. Syst. Model. 3 (3) (2004) 235–253.
[27] M. Brandozzi, D.E. Perry, Transforming goal-oriented requirement specifications into architecture prescriptions, in: ICSE 2001 STRAW Workshop, 2001.
[28] M. Brandozzi, D.E. Perry, From goal-oriented requirements to architectural prescriptions: The preskriptor process., in: STRAW, Citeseer, 2003, pp. 107–113.
[29] C. Hofmeister, R.L. Nord, D. Soni, Global analysis: moving from software requirements specification to structural views of the software architecture, IEE Proc. Softw. 152 (4) (2005) 187–197.
[30] A. Van Lamsweerde, From system goals to software architecture, in: International School on Formal Methods for the Design of Computer, Communication and Software Systems, Springer, 2003, pp. 25–43.
[31] J.G. Hall, M. Jackson, R.C. Laney, B. Nuseibeh, L. Rapanotti, Relating software requirements and architectures using problem frames, in: Proceedings of the IEEE Joint International Conference on Requirements Engineering, 2002, pp. 137–144.
[32] M. Jackson, Problem Frames: Analysing and Structuring Software Development Problems, Addison-Wesley, 2001.
[33] P. Bourque, R.E. Fairley, et al., Guide to the software engineering body of knowledge (SWEBOK): Version 3.0, IEEE Computer Society Press, 2014.
[34] D.L. Parnas, P.C. Clements, D.M. Weiss, The modular structure of complex systems, IEEE Trans. Softw. Eng. 3 (1985) 259–266.
[35] S. Apel, C. Kästner, An overview of feature-oriented software development, J. Object Technol. 8 (5) (2009) 49–84.
[36] R.N. Langlois, Modularity in technology and organization, J. Econ. Behav. Organ. 49 (1) (2002) 19–37.
[37] R. Davis, Business Process Modelling with ARIS: A Practical Guide, Springer Science & Business Media, 2001.
[38] H.A. Reijers, J. Mendling, R.M. Dijkman, Human and automatic modularizations of process models to enhance their comprehension, Inf. Syst. 36 (5) (2011) 881–897.
[39] S. Brinkkemper, S. Pachidi, Functional architecture modeling for the software product industry, in: Proceedings of the European Conference on Software Architecture, 2010, pp. 198–213.
[40] F.B. e Abreu, M. Goulão, Coupling and cohesion as modularization drivers: Are we being over-persuaded ? in: Proceedings of the European Conference on Software Maintenance and Reengineering, 2001, pp. 47–57.
[41] O. Liskin, R. Pham, S. Kiesling, K. Schneider, Why we need a granularity concept for user stories, in: Proceedings of the International Conference on Agile Software Development, Springer, 2014, pp. 110–125.
[42] S. Espana, N. Condori-Fernandez, A. Gonzalez, Ó. Pastor, Evaluating the completeness and granularity of functional requirements specifications: A controlled experiment, in: Proceedings of the IEEE International Requirements Engineering Conference, 2009, pp. 161–170.
[43] C. Kästner, S. Apel, M. Kuhlemann, Granularity in software product lines, in: Proceedings of the International Conference on Software Engineering, 2008, pp. 311–320.
[44] J.D. McKeen, H. Smith, Making IT happen: Critical Issues in IT Management, Wiley Chichester, 2003.
[45] J. Cleland-Huang, O.C. Gotel, J. Huffman Hayes, P. Mäder, A. Zisman, Software traceability: Trends and future directions, in: Proceedings of Future of Software Engineering, 2014, pp. 55–69.

[46] O. Gotel, J. Cleland-Huang, J.H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, G. Antoniol, The quest for ubiquity: A roadmap for software and systems traceability research, in: Proceedings of the IEEE International Requirements Engineering Conference, pp. 71–80.
[47] A. Egyed, P. Grünbacher, Automating requirements traceability: Beyond the record & replay paradigm, in: Proceedings of the International Conference on Automated Software Engineering, 2002, pp. 163–171.
[48] Y. Zhang, R. Witte, J. Rilling, V. Haarslev, An ontology-based approach for traceability recovery, in: Proceedings of the International Workshop on Metamodels, Schemas, Grammars and Ontologies, 2006, pp. 36–43.
[49] M. Rath, J. Rendall, J.L.C. Guo, J. Cleland-Huang, P. Mäder, Traceability in the wild: Automatically augmenting incomplete trace links, in: Proceedings of the International Conference on Software Engineering, 2018, pp. 834–845.
[50] M. Borg, P. Runeson, A. Ardö, Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability, Empir. Softw. Eng. 19 (6) (2014) 1565–1616.
[51] A. Tang, P. Liang, V. Clerc, H. Van Vliet, Traceability in the co-evolution of architectural requirements and design, in: Relating Software Requirements and Architectures, Springer, 2011, pp. 35–60.
[52] P. Rempel, P. Mäder, Estimating the implementation risk of requirements in agile software development projects with traceability metrics, in: Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality, 2015, pp. 81–97.
[53] S.L. Pfleeger, S.A. Bohner, A framework for software maintenance metrics, in: Proceedings of the International Conference on Software Maintenance, 1990, pp. 320–327.
[54] A. Murugesan, S. Rayadurgam, M. Heimdahl, Requirements reference models revisited: Accommodating hierarchy in system design, in: Proceedings of the IEEE International Requirements Engineering Conference, 2019, pp. 177–186.
[55] T. Sharma, D. Spinellis, A survey on software smells, J. Syst. Softw. 138 (2018) 158–173.
[56] G. van Valkenhoef, T. Tervonen, B. de Brock, D. Postmus, Quantitative release planning in extreme programming, Inf. Softw. Technol. 53 (11) (2011) 1227–1235.
[57] K. Vlaanderen, S. Jansen, S. Brinkkemper, E. Jaspers, The agile requirements refinery: Applying SCRUM principles to software product management, Inf. Softw. Technol. 53 (1) (2011) 58–70.
[58] N. Ali, S. Baker, R. O'Crowley, S. Herold, J. Buckley, Architecture consistency: State of the practice, challenges and requirements, Empir. Softw. Eng. 23 (1) (2018) 224–258.
[59] G. Lucassen, M. van de Keuken, F. Dalpiaz, S. Brinkkemper, G.W. Sloof, J. Schlingmann, Jobs-to-be-done oriented requirements engineering: a method for defining job stories, in: Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality, 2018, pp. 227–243.
[60] I. Inayat, S.S. Salim, S. Marczak, M. Daneva, S. Shamshirband, A systematic literature review on agile requirements engineering practices and challenges, Comput. Hum. Behav. 51 (2015) 915–929.
[61] G. Lucassen, F. Dalpiaz, J.M.E. van der Werf, S. Brinkkemper, Improving agile requirements: The Quality User Story framework and tool, Requir. Eng. 21 (3) (2016) 383–403.
[62] Y. Wautelet, S. Heng, M. Kolp, I. Mirbel, S. Poelmans, Building a rationale diagram for evaluating user story sets, in: Proceedings of the International Conference on Research Challenges in Information Science, 2016, pp. 1–12.
[63] E.S. Yu, Towards modelling and reasoning support for early-phase requirements engineering, in: Proceedings of the International Symposium on Requirements Engineering, 1997, pp. 226–235.
[64] J. Bosch, Software architecture: The next step, in: Proceedings of the European Workshop on Software Architecture, 2004, pp. 194–199.
[65] A. Hubaux, T.T. Tun, P. Heymans, Separation of concerns in feature diagram languages: A systematic survey, ACM Comput. Surv. 45 (4) (2013).
[66] J. Garcia, D. Popescu, G. Edwards, N. Medvidovic, Identifying architectural bad smells, in: Proceedings of the European Conference on Software Maintenance and Reengineering, IEEE, 2009, pp. 255–258.
[67] S. Smirnov, R. Dijkman, J. Mendling, M. Weske, Meronymy-based aggregation of activities in business process models, in: Proceedings of the International Conference on Conceptual Modeling, Springer, 2010, pp. 1–14.
[68] A. Bucchiarone, S. Gnesi, P. Pierini, Quality analysis of NL requirements: An industrial case study, in: Proceedings of the IEEE International Conference on Requirements Engineering, 2005, pp. 390–394.
[69] D.M. Berry, Evaluation of tools for hairy requirements and software engineering tasks, in: Procedings of the Workshop on Empirical Requirements Engineering, 2017, pp. 284–291.
[70] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Science & Business Media, 2012.
[71] T.C. Lethbridge, S.E. Sim, J. Singer, Studying software engineers: Data collection techniques for software field studies, Empir. Softw. Eng. 10 (3) (2005) 311–341.
[72] J. Bosch, Achieving simplicity with the three-layer product model, Computer 46 (11) (2013) 34–39.
[73] A. Gonzalez, S. Espana, O. Pastor, Unity criteria for business process modelling, in: Proceedings of the International Conference on Research Challenges in Information Science, 2009, pp. 155-164.