b
UNIVERSITÄT
BERN

Modeling requirements artifacts in an IDE

Master's Thesis

Robert Niemiec

from

University of Bern

Faculty of Science, University of Bern

1st September 2020

Prof. Dr. Oscar Nierstrasz

Nitish Patkar & Andrei Chiş & Nataliia Stulova

Software Composition Group

Institute of Computer Science

University of Bern, Switzerland

Abstract

A plethora of artifacts are available for practitioners to document and communicate requirements. Artifacts evolve as the requirements engineering (RE) process advances. Therefore, using numerous artifacts with various granularities of details they hold, causes problems like manual translation efforts and inconsistencies while reflecting requirements changes across artifacts. Furthermore, employing distinct tools for creation and management of artifacts only leads to fragmented knowledge, and eventually, traceability issues.

In this thesis, we propose an approach to manage requirements related artifacts inside a single platform, *i.e.*, the integrated development environment (IDE). First, we compiled a list of artifacts from a selection of studies. We performed a classification of the selected artifacts to reason about their characteristics and usage patterns. Towards providing a practical tool to illustrate our approach, we modeled a selection of artifacts in the Pharo programming environment. It serves as a centralized platform for creating, viewing, and managing requirements through a combination of selected artifacts.

We speculate that such an approach can bring requirements specification closer to their implementation. Additionally, it will facilitate and support collaboration among diverse stakeholders by preserving their knowledge within a single tool.

Contents

1 Introduction

Requirements play a central role in software development activities [13]. They specify the desired functionality and non-functional attributes of the product-to-be. Clearly expressed requirements act as a contract between project stakeholders and mitigate misunderstandings between them. Therefore, requirements must be thoroughly discussed before they are accepted for implementation. The communication of requirements is, however, a challenging task because project stakeholders vary in many aspects, such as their technical expertise, domain knowledge, or preferred methods of communication [1]. To support discussions, stakeholders employ several formats of requirements, such as *user stories* and related artifacts, such as *Kanban boards* at different phases of software development. Traditional waterfall methodologies focus on strict documentation, such as extensive specifications and reports. On the other hand, agile processes prefer methods that facilitate close interaction between stakeholders, *e.g.*, frequent meetings, and individual interactions [13]. Consequently, clients and other stakeholders must participate in these meetings to agree upon a definitive version of the requirements. However, there is no definite way of involving stakeholders in agile processes [16].

The broad nature of the term *artifact* has led to a multitude of requirements formats and corresponding artifacts being used for distinct purposes [7, 9], each with their own merits and limitations [9]. Often, artifacts are distributed among the tools that facilitate their creation or manipulation. Similarly, they get lost in media of communication, *e.g.*, emails, and physical objects, for example, sticky notes [9, 20]. Such dispersion of requirements leads to fragmented knowledge and translates to challenges in

requirements traceability. Frequent requirements changes, subsequently, demand dynamic management of requirements [2, 16]. Furthermore, artifacts are used in several phases of software development for distinct purposes. For instance, user stories are suitable during planning and monitoring, but they seldom convey or efficiently document the overall vision of the project [9]. Similarly, generic documents are useful for requirements collection, since they offer an easy way to add information, but they can be difficult to search, making them sub-optimal for management tasks [9]. As a consequence, identifying the right combinations of artifacts that enhance collaboration among technical and non-technical stakeholders is challenging.

As the requirements engineering (RE) process proceeds, artifacts are subject to change [14]. For example, certain requirements can be modified to adapt to evolving customer demands, and others removed because of time constraints. Thus, they have to be maintained and monitored, *i.e.*, reflecting requirements changes across all associated artifacts and checking for inconsistencies [9, 20]. This can be challenging in a distributed requirements environment, where different documentation and management tools can have interconnected artifacts, *e.g.*, using physical documents, spreadsheets for documentation, and an RE tool like IBM Rational DOORS for management. In such a scenario, mistakes related to consistent updating of changes may happen, causing subsequent inconsistencies and miscommunication, *e.g.*, forgetting to update a requirement across tools and documents. Additionally, a complete lack of communication is possible in certain situations. Different stakeholders may elect to ignore certain artifacts, which may contain vital information for understanding other aspects of the project. For instance, developers sometimes do not pay sufficient attention to design-related artifacts, such as personas, and only discuss user experience-related issues on-demand with design specialists. This increases the likelihood of developers not gaining a comprehensive understanding of the target users, even though the product is being designed with them in mind. In summary, different types of stakeholders may use distinct artifacts and tools for their work, leading them to possibly neglect others.

To address the issues emerging from the dispersion of requirements artifacts across tools and media, we propose an approach to model and manage software requirements within a single platform. With our approach, we aim to streamline the requirements engineering process for all project stakeholders. It assists stakeholders to model requirements artifacts in a particular sequence, i.e., from high-level to more specific ones. Additionally, stakeholders can also model any additional artifacts or mold the existing ones to suit their preferences. Our approach encourages stakeholders to link artifacts closely with the corresponding implementation to facilitate smooth navigation. With the help of several views, stakeholders can monitor the evolving requirements and the corresponding implementation. We believe that such an approach could help to facilitate requirements discussion and management across all software engineering (SE) phases. Furthermore, we speculate that if requirements are kept closer to their implementation, then tasks such as traceability and documentation would be less cumbersome. As a research output, we introduce a tool that assists stakeholders to model and maintain project requirements in one place.

In this thesis, we aim to answer the following research questions:

- RQ_1: What are the most commonly used requirements formats and related artifacts within the project development process?

We analyzed a selection of related publications to compile a list of artifacts.

- RQ_2: What are the main characteristics of these artifacts?
  We propose classification schemes to reason about artifact characteristics and their propagation into different development phases.
- RQ_3: What advantages do we gain if we specify and manage artifacts within a single platform?
  We discuss our approach and prototype implementation that allows stakeholders to model requirements within an IDE.

The remainder of the thesis is structured as follows: In chapter 2, we report related work and outline limitations of current requirements engineering tools, chapter 3 presents the list of identified artifacts, proposes their classification, and discusses the findings. In chapter 4, we present an approach and prototype implementation to specify and manage requirements in a single platform. In chapter 5, we discuss the main features and supported workflows in the prototype implementation. chapter 6 discusses advantages of our approach over the existing ones, and finally, chapter 7 concludes our principle contributions.

2

Related work

Artifacts are fundamental enablers of the RE process. Several artifacts are created across the project development lifecycle. Their vast scope warrants further analysis of their properties. Although numerous studies discuss requirements artifacts, most of them did not study their characteristics, nor did they perform a classification of them. This chapter reports previous attempts to compile artifacts lists and limitations of current requirements management tools.

RE studies without artifacts classification. Several studies report which artifacts are used to collect and document the requirements. However, most of them only mention the artifacts and fail to perform their comprehensive classification. We analyze these studies to compile a list of artifacts. It is then used in our classification to identify their characteristics.

Schoen *et al.* conducted an SLR to study stakeholder and end-user involvement in agile RE practices [16]. They identified that agile practices are leveraging different research fields, such as user-centered design and software engineering. They compiled a list of 57 artifacts mentioned in the analyzed studies. Additionally, they discuss the role of artifacts used to document requirements in requirements management. Since the artifacts are pooled from various RE studies, we reckon that this list provides a good overview of them. Consequently, it is used in our classification.

Another SLR by Schoen *et al.* focuses on agile RE patterns in industry [17]. They identified 14 requirements artifacts mentioned in the analyzed papers. They matched 14 artifacts with identified agile

RE problems that they address. Some of the problems include difficulties in iterative involvement of stakeholders, refining requirements collaboratively, and maintaining technical or functional dependencies to other project teams. As with the previous paper, this publication extracts artifacts mentioned in several RE studies, which adds to the overview of used artifacts. Hence, we use a selection of them in our classification.

A study by Bass discusses artifacts usage in large-scale offshore software development programs [2]. Specifically, artifacts were analyzed in terms of how they map to different software development processes, *e.g.*, sprints. They identified 25 artifacts that were considered significant, along with a few others that were only briefly discussed. The main 25 artifacts were organized into five categories of abstraction, such as release and sprint. These categories correspond to project planning aspects. However, the study does not analyze those artifacts in terms of their other characteristics. Nevertheless, we found several artifacts that are unique to the field of large-scale agile development. Therefore, we used a subset of these artifacts in our classification.

RE studies with artifacts classification. It is important to study artifacts in terms of their characteristics to identify their key attributes and features. Consequently, few other studies performed classifications of artifacts.

Particularly, Liskin performed an interview study with the aim to understand how requirements artifacts are used in practice, *i.e.*, which activities they support, and what challenges working with multiple artifacts concurrently poses [9]. Part of their work was a classification scheme for the artifacts mentioned by the interviewees. The scheme focuses on whether the artifact is a container, how user-oriented it is, and whether it is a solution model. It is difficult to discern from this study when the artifacts are created and consumed. Nevertheless, we took a subset of the mentioned artifacts and classified them according to our custom scheme.

Another paper by Liskin *et al.* discusses requirements artifacts in terms of granularity, *i.e.*, the scope of a requirement [10]. They discuss the effects of artifact vagueness on their implementation. However, the study only looked at a single software project, which makes the results not generalizable.

The existing studies fail to provide an extensive overview of the characteristics of the involved artifacts. To tackle this, we propose a custom classification scheme in chapter 3. With our classification, we intend to broaden the understanding of artifacts use in various SDLC phases. Similarly, we aim to discuss various characteristics, such as artifact formats and flow. The overall motivation is to discover the complexities of requirements management arising due to the studied characteristics of different artifacts.

Requirements Management Tools. There are several requirements engineering tools that attempt to integrate artifacts into the project development process. Most of them focus on specific features and stakeholders.

The ProR tool from the Eclipse Requirements Management Framework (RMF) is a tool for managing

requirements within the Eclipse IDE.[1] It supports storing and updating requirements from IDE. Numerous formats, such as ReqIF are supported. Additionally, ProR includes a feature to manually link requirements to each other, *e.g.*, creating nested requirements. For instance, users can select specific requirements and add child elements to them. These elements are of the same type, but with a different relation to each other, *i.e.*, one is a parent, the other is a child object. However, ProR lacks visualization support, which makes getting an overview of the requirements structure and other aspects, such as work progress, more difficult. Also, there is no modeling support, *i.e.*, stakeholders cannot create new requirement artifacts, thus having to rely solely on the infrastructure provided by ProR. Finally, to the best of our knowledge, ProR does not provide a feature to link code entities, *e.g.*, classes, to the requirements within ProR itself.

Enterprise Architect (EA) by Sparx Systems is another tool that serves as a requirement modeling and management platform.[2] Unlike ProR, EA is a standalone application and it is not integrated into an IDE. It provides features for managing and visualizing requirements. Project management is supported through the assignment of stakeholders to artifacts to estimate project size and complexity. A built-in source code editor allows users to navigate between requirements models and their source code. Requirements can also be displayed visually using a range of built-in model templates, *e.g.*, diagrams can be used to illustrate business models. Finally, the EA enables source code generation from requirements models, along with customizing the generation process to fit company and industry standards. EA provides features for aiding stakeholders across the development lifecycle. However, there is no support for simple requirements overviews, *i.e.*, the range of provided artifacts does not enable stakeholders to get an efficient overview of the requirements structure of a project. Also, there is no support to model custom requirements artifacts. EA is a standalone tool, which forces users to conform to its standards and requirements artifacts. This lack of flexibility is a limiting factor for project development teams, in case they require different artifacts.

Rational DOORS by IBM falls into the same category of feature-rich RE tools like Enterprise Architect.[3] It provides various features for the management of requirements. Specifically, linking is supported by linking table data across different modules of the tool. It also centralizes the RE process by storing requirements in one platform. Finally, the tool ensures the compliance of a modeled solution to the requirements, *i.e.*, the system analyzes if the requirements are met by the models devised by the stakeholders. However, DOORS suffers from a few limitations. Like EA, there is a lack of flexibility in terms of modeling new artifacts, which forces users to utilize the structure given by DOORS. Also, the breadth of implemented features makes it difficult to get a quick overview of the available functionalities.

Although the above tools provide useful features, there are few important limitations. For example, most requirements tools offer good support for most RE activities; however, they do not fully support the modeling and management of requirements. For instance, these tools often do not use consistent data formats for artifacts. This constitutes a challenge in the consistent management of requirements. Also, there is a lack of support for modeling custom artifacts. Finally, most tools do not provide features for getting a visual overview of the requirements, *e.g.*, their structure, and dependencies.

[1] `https://www.eclipse.org/rmf/`
[2] `https://sparxsystems.com/products/ea/index.html`
[3] `https://www.ibm.com/products/requirements-management`

3

Artifacts and their characteristics

Answering our first two research questions requires an analysis of requirements engineering (RE) literature to gather information about artifacts used in software projects. One possibility is to carry out a large scale systematic literature review (SLR) to gain a thorough overview of the RE processes and artifacts used therein. However, this approach presents a few issues in the context of this thesis.

The vague nature of the term *artifact* results in challenges regarding the appropriate construction of the search terms that are relevant to our work. Searching for "requirements artifacts" on Google Scholar returns tens of thousands of results. Upon inspection, the papers cover a large variety of subjects, *e.g.*, machine learning, visualizations, and methodologies. In many of these papers, requirements artifacts are often mentioned in passing and do not focus on that topic. Hence, it is challenging to manually analyze and obtain results suitable for the purposes of artifacts classification. Finally, our work involves the implementation of a practical approach to model artifacts in addition to a theoretical framework. Thus, time constraints prevent us from conducting a full-scale SLR.

In light of these issues, we decided to look only at a selection of publications [2, 9, 12, 16, 17]. The publications were chosen based on their focus on describing and classifying requirements artifacts. We found that most RE papers mention artifacts in passing, or only discuss a small number of them. Our focus is to compile an extensive list of artifacts. Consequently, the publications we chose analyze a number of artifacts that we deem significant, *i.e.*, at least 10. Additionally, one of the publications includes the results of an online survey related to artifacts used by practitioners [12]. The number of results therein

Table 3.1: Dimensions for the artifacts classification

Dimension	Possible values
Format	Textual, Graphical, Mixed
Nature	Digital, Physical,
Contains	*other artifacts*
Helps create	*other artifacts*
SDLC phase(s) of origin	Requirements, Design, Development and Testing, Deployment and Maintenance
SDLC phase(s) of use	Requirements, Design, Development and Testing, Deployment and Maintenance

significantly increases the size of our list.

From the above papers, we extracted the referenced artifacts and classified them along several dimensions as explained in section 3.1.

3.1 Artifact classification

In order to compile a list of artifacts, we: (1) searched through the selected publications for mentions of artifacts, and (2) removed any duplicates. After completion of these steps, we compiled a list of 89 artifacts. Through discussion, items deemed not related to requirements engineering were filtered out. Eventually, we obtained a list of 62 artifacts, as provided in Table A.1. The first column specifies the artifact, whereas the next three columns denote whether the corresponding artifact is physical or digital, and its format, *i.e.*, graphical, textual, or mixed.

The selected artifacts were classified along 6 dimensions as presented in Table 3.1. The first column denotes the dimension followed by possible values.

The dimension “Format” describes the format of a given artifact, *i.e.*, graphical, textual, or a combination of both. The dimension “Nature” denotes whether the artifact is physical or digital. The dimension “Contains” specifies if the artifact can function as a container, *e.g.*, a story map, which contains story cards. The dimension “Helps create” pertains to the observation that some artifacts can be used as an intermediate step towards the creation of another. This dimension lists potential artifacts that can be constructed with the help of the original one. For example, sketches help create UI wireframes.

Finally, the last two dimensions correspond to the software development life cycle (SDLC) phases where a specific artifact originates and is subsequently consumed. Since there are numerous SDLC conventions [11, 15, 18], a specific model needed to be agreed upon for consistent classification. Consequently, we considered the following phases:

- Requirements— where requirements are elicited, collected and specified.

- Design— where the set of collected requirements is reasoned about, and design of the solution to be developed is created. This phase also includes the structuring of a workflow scheme for the actual development and delivery of the product. One can think of the design process also entailing the design of the development process itself.
- Development and Testing— where the solution is developed and source code is created and tested.
- Deployment and Maintenance— where the solution is delivered to the client and deployed in a real-world scenario. Also, the solution is maintained, which entails handling bug fixing and communication with the customers.

The dimension "SDLC phase(s) of origin" specifies the phase in which the artifact is usually created. The dimension "SDLC phase(s) of use" specifies the phases in which the artifact is usually consumed.

Using the above dimensions and list of artifacts, we performed our classification. The results are summarized below.

Formats. Of the 62 analyzed artifacts, 16 (26%) are textual, *e.g.*, user stories. A further 18 artifacts (29%) have a graphical format, for example, sketches. The remaining 28 elements (45%) are classified as having both textual and graphical properties, *e.g.*, prototypes.

Nature. The vast majority of artifacts, *i.e.*, 94%, are digital, *e.g.*, data models and use cases. 16 artifacts (26%) are physical in nature, for instance, story cards. Most of them seem to be related to workflow monitoring and meeting activities, *e.g.*, status boards and index cards. 13 (21%) artifacts, such as prototypes, come in both digital and physical forms.

Contains. Table A.2 summarizes our results to show which artifacts contain other artifacts. The first column denotes the host artifact, whereas the remaining columns denote the contained artifacts. A '✓' is used in places where an artifact is contained in the host artifact. 19 of the 62 artifacts (31%) are used as containers, *i.e.*, they can contain at least one other type of artifact from the list. Examples of containers are story boards, which contain story cards, and release plans, which contain features.

Helps create. Out of the 62 artifacts, 31 (50%) can help others be created. For example, sketches can help guide the design of wireframes and design concepts.

Flow across phases. To illustrate the flow of artifacts from the phases of origin to phases of use, we present a Sankey diagram that contains three columns, see Figure 3.1. The left and right columns indicate the SDLC phases of origin and use, respectively. The height of the nodes in these columns signifies their frequency of occurring, *i.e.*, the more artifacts are involved in a phase, the larger its respective node. The middle column contains the classified artifacts. The height of their nodes also indicates if they are involved in multiple phases, *i.e.*, the more phases an artifact is involved in, the larger its respective node. Each artifact is connected with at least one node from the left and right columns. The connections are

color-coded according to the distinct phases.

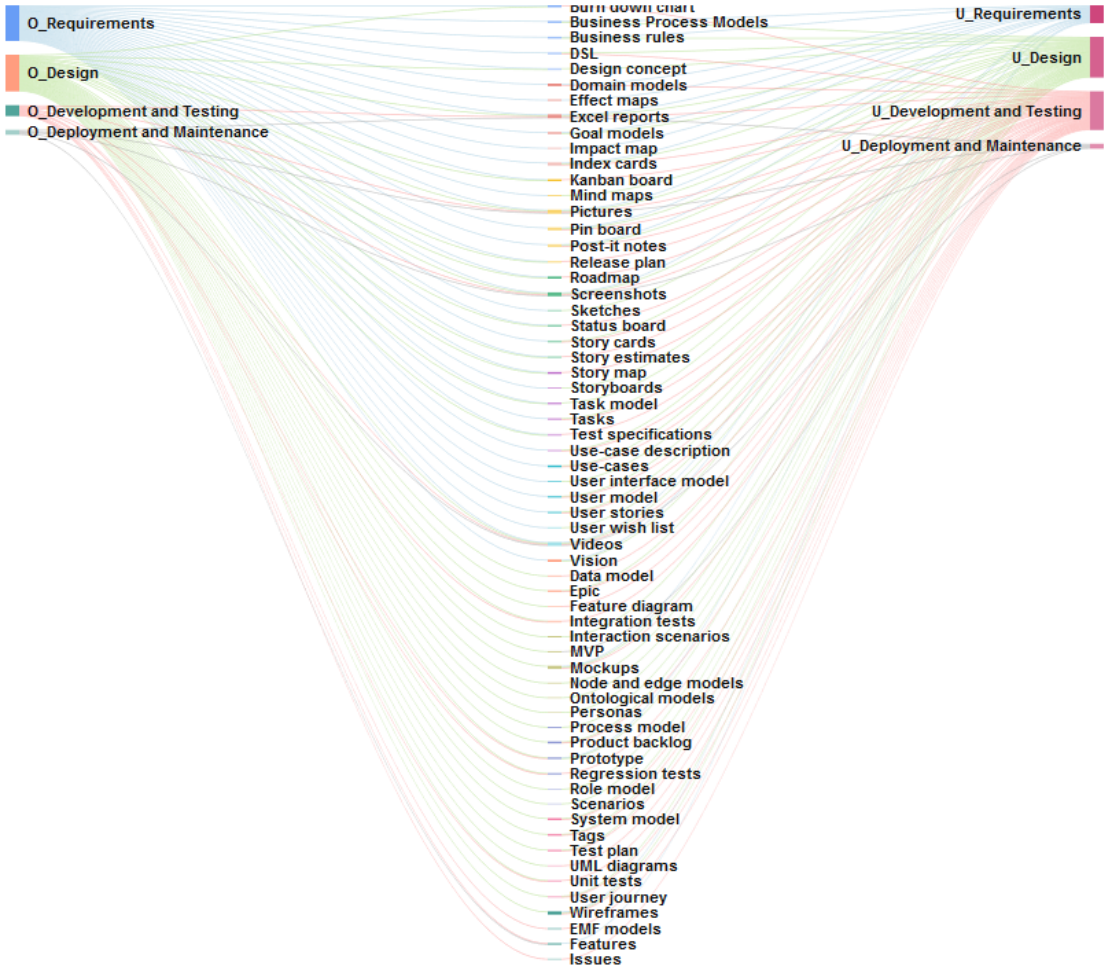

Figure 3.1: The flow of artifacts along SE phases

Concerning the phases of origin, we observed that 36 out of 62 artifacts (58%) originate in the requirements phase, 37 (60%) originate in the design, 11 (18%) originate during the development and testing, and only 5 (8%) originate during the deployment and maintenance phase. Of the artifacts originating in the first two phases, most are then used for design and development purposes.

Regarding the phases of use, 18 of 62 artifacts (29%) are used in the requirements phase, 41 (66%) are consumed during the design, 39 (63%) in development and testing phase, and only 5 artifacts (8%) in the deployment and maintenance phase.

3.2 Discussion

From the above results, we derived a set of findings concerning the trends and characteristics of the analyzed artifacts. Below, we discuss the important findings along with potential solutions.

A plethora of artifacts, with varying characteristics, are available to practitioners; these artifacts need to be easily accessible to stakeholders

The elicitation and management of requirements with a diverse group of stakeholders, often with varying goals and technical skills, result in a plethora of artifacts [9, 16]. The existence of different development approaches and methodologies (*e.g.*, user-centered design, scrum) further contributes to the diverse collection of artifacts [6, 16]. The selection of artifacts is also influenced by the wide range of activities that they are meant to aid, *e.g.*, designing the solution or monitoring development workflow.

We observed that artifacts are created in all four SDLC phases. This might be happening because artifacts may be created in different tools and on different mediums, depending on the activities they intend to support. For instance, a sketch might be created on a piece of paper to be used as a discussion medium in a design meeting. Other artifacts, such as test plans, are mostly programmed in different digital tools, *e.g.*, IBM DOORS or JIRA, and are helpful for planning development activities and monitoring test progression. This dispersion of artifacts may cause stakeholders to neglect certain artifacts [3].

Developers need a complete picture of the requirements structure since they implement the final system. A better comprehension of the "big picture" is needed to produce a product that satisfies the stakeholder vision. We posit that integrating artifacts in one platform and making them accessible to all stakeholders, can contribute to better communication and clarification of requirements. For instance, developers can check the different artifacts within the tool and try to clarify any uncertainties with other stakeholders rather than ignoring some requirements and focusing purely on implementation.

If artifacts are meant to be used by developers and designers, then they should be clear for these groups

Buchan studied requirements understanding, particularly, the impact of communication among stakeholders on the shared understanding of the final product [4]. He identified the divergence of stakeholders, as well as the plethora of used artifacts as challenges for achieving a shared understanding of requirements. Consequently, he proposed solutions, such as creating sufficient opportunities to discuss and identify gaps in shared understanding through frequent stakeholder meetings. Similarly, Khan *et al.* reported issues in requirements communication and management [8]. They identified non-formal and vague documentation of requirements to be a distracting factor for developers and designers. They also pointed out requirement changes as being a critical step in software development processes. However, they noted that proper management of such changes has been an open issue for decades.

A significant number of artifacts originate in the requirements (58%) and design (60%) phases, as a result of documenting and structuring requirements through discussions. Most artifacts are used for design (66%), as well as development and testing purposes (63%). This is to be expected since the intended audience for requirements is the designers and developers tasked with creating the solution. It falls upon these two groups to reason about these requirements, and develop a solution that ultimately satisfies the customers. Frequent discussion of requirements is necessary to uncover any gaps in a shared understanding between stakeholders. Furthermore, non-formal documentation can be confusing for design and development. Consequently, steps need to be taken to apply structure to requirements, *i.e.*, break down vague requirements into actionable steps for developers.

Putting artifacts inside one tool creates a convenient environment for the joint exploration of artifacts. It is important to make the full requirements structure accessible and understandable to developers and designers. The solution could be used as a platform for discussion between stakeholders. During meetings, structured requirements can be created immediately in the tool, and discussed in real time. Developers and designers can clarify and adjust the requirements to suit their needs, *i.e.*, achieve a more technical view of the requirements. This would facilitate striving towards a complete picture of a project's requirements, shared by all stakeholders.

Reflecting requirements changes across artifacts can be challenging

Maintenance activities can significantly influence a project's trajectory, both in functionality and development scheduling [19]. Changes can occur in the forms of additions, changes to existing requirements, and deletions. The way that stakeholders adapt to these changes and apply them is crucial to continuous customer satisfaction.

In most cases, the analyzed artifacts are distributed among different platforms, *e.g.*, user stories can be stored in a document and on story cards. Only 5 analyzed artifacts are used in the last phase of our proposed SDLC. To facilitate the efficient application of requirements changes in software, they have to be communicated to and understood by all stakeholders. Changes usually occur suddenly, which requires stakeholders to respond to these changes as quickly as possible. Moreover, changes have to be documented consistently, across all media where they are present [9, 20].

It stands to reason that enabling project requirements to be managed in a centralized way can help facilitate requirements changes. By integrating artifacts inside a single platform, the changes would be more immediately noticeable for the developers, and could subsequently be addressed with other stakeholders. Also, the additional effort needed to reflect changes across multiple mediums would be alleviated by using a single RE tool. We posit that maintaining artifacts within a single platform may help to maintain artifact freshness after deployment. Updates are also easier if they are integrated into a central environment. As an example, consider goal models, with each goal carrying references to objects which work towards achieving that goal. As the product is released and maintained, the goal model can be edited: new goals can be added, and existing ones changed. This way, the RE process is more easily maintained.

Artifacts have relations and dependencies; gaining an overview of the structure is important for understanding the complete requirements picture

Artifacts often exist in hierarchies and have dependencies with each other. Liskin found that linking artifacts is a common way to relate requirements and navigate between them [9]. Several methods for linking artifacts exist, such as manual referencing, attachment, *i.e.*, using containers, creating a distinct link, and using intermediary artifacts [9]. These methods were found to have several drawbacks, such as the effort needed to perform linking, and lack of clear guidelines. This makes working with multiple artifacts challenging.

As different artifacts are being created and maintained in a software project, the importance of keeping them well-organized increases. Specifically, relations and dependencies between artifacts have to be clearly visible. Otherwise, stakeholders may miss crucial information when looking through the requirements. We found that half of the analyzed artifacts can help to create other artifacts. As an example, epics can be refined into use cases, which can in turn be split into related user stories.

In this case, it seems useful to provide simple linking functionality for artifacts. These links should be visible and apparent upon looking at the requirements structure. An approach to formalize the relations between artifacts also seems promising. For example, establishing a hierarchy which puts epics at the highest level, and allows use cases to be created and automatically associated with said epic, reduces linking efforts. The perfect environment for imposing this structure is a programming tool, where practitioners have to adhere to the constraints set by said tool. We posit that this approach would streamline artifacts management processes considerably.

3.3 Summary

Our findings from the classification, as well as our research on RE tools, highlighted a set of challenges in requirements management, namely:

- The more distributed the requirements are, the harder it is to gain a perspective on their overall structure.
- A significant amount of effort is needed to reflect changes across all artifacts in distributed settings.
- Stakeholders need to be able to gain insight into the various dependencies of artifacts.
- RE tools have inadequate support for requirements management and modeling.
- Visualization capabilities are not adequately leveraged by most RE tools, even though they help establish concepts, such as requirements structure, in a more approachable manner.

These challenges motivate our idea of creating an integrated requirements workflow in development tools, as discussed in the next chapter.

4

Moldable Requirements Manager

As discussed earlier, the dispersion of requirements presents risks and challenges for management during the development lifecycle. First, the distribution of artifacts across multiple media and tools creates challenges for focusing on the "big picture" of a project. Second, since developers are the intended audience for requirements, they need to have easy access to requirements to be able to clarify them. Third, as requirements change over time, it is difficult to reflect their changes across all artifacts, especially in a dispersed requirements environment. Finally, artifacts often have dependencies and relations, which have to be visible. This can be challenging in cases where objects have links across different tools and media.

To address these issues, we outline our approach to integrate requirements workflow within a single platform. We designed a platform for requirements creation and management. Consequently, we present our prototype implementation, "Moldable Requirements Manager" (MReM) that manifests the principle ideas behind our approach. The platform includes a graphical interface for managing project requirements more easily, *i.e.*, addition, updating, and deleting artifacts. We also use visualization techniques to more eloquently present aspects such as requirements structure and work progress.

MReM is intended to be used as a communication platform during stakeholder meetings. We intend to encourage discussion and clarification of artifacts during all the phases of SDLC so that the developers have a clear picture of what to implement.

4.1 Design philosophy

One of the main challenges in requirements management is that artifacts are scattered across several media and tools. For example, design patterns are documented in terms of personas and scenarios, which are typically kept in a shared Word document. Workflow artifacts, such as status boards and story cards, are kept in physical form. Finally, popular agile requirements formats, such as user stories and epics, are often maintained in a web tool for management. This dispersion can lead to issues such as: (1) negligence of certain artifacts by some stakeholders, *e.g.*, personas being neglected by developers, (2) more effort needed to reflect requirement changes across all artifacts, and (3) challenges in linking artifacts, especially if they are in different tools and media.

Upon looking at these issues, it seems apparent that a different approach to requirements management is needed. The principal ideas behind our approach are that: (1) requirements are elicited and modeled within a single tool, (2) requirements can be monitored using visualizations, such as mind maps, (3) hierarchical decomposition (*e.g.*, use cases into user stories) through simple linking structures eases requirements navigation, and (4) a simple user interface (UI) enables non-technical stakeholders to participate in the RE process. The idea of centralized RE addresses the issue of possible negligence of certain artifacts. By keeping artifacts easily accessible in one place, stakeholders can get a better overview of the full requirements structure. Visualizations enable stakeholders to get an overview of critical project aspects, such as the artifacts structure and workflow monitoring. The hierarchical decomposition enables the easy establishment of artifacts structures and helps maintain them consistently. Finally, non-technical stakeholders can be encouraged to participate more actively in RE processes through the user-friendly UI.

The main application of our approach is within a software development environment, where: (1) different stakeholders hold requirements meetings, (2) requirements change frequently and need to be documented appropriately, and (3) monitoring of development workflow is needed to ensure the on-time delivery of the product to customers.

4.1.1 Infrastructure

MReM aims to provide appropriate features to support the above founding ideas. It is built on top of a Pharo-based GToolkit environment.[1]

Pharo. Pharo is a modern dynamic programming language, that stems from Smalltalk. Pharo also offers a separate IDE. The structure of the language and Pharo Virtual Machine then allows developers, similar to Javascript, to modify and execute the code on the fly.

GToolkit. GToolkit is a moldable development environment that aims to substitute a standard coding experience. It improves on existing Pharo tools such as Inspector and Coder interfaces. The core idea of GToolkit is to integrate numerous SE activities into a single tool. This makes GToolkit an excellent

[1] `https://gtoolkit.com/`

candidate to implement our approach since we simply extend the core idea to the requirements engineering realm.

4.1.2 Building blocks

We leverage several functionalities of the Pharo and GToolkit environments to implement our tool.

Modeling capabilities. The GToolkit environment enables the creation of domain objects and model workflows. This allows us to craft different requirements artifacts as regular objects and customize them as needed. We use these modeling features to implement a selection of requirements artifacts and establish their hierarchy. Consequently, MReM follows a pre-defined artifact structure as discussed in subsection 4.2.1.

Moldable visualization. One of the main features of GToolkit is its elaborate visualization system. Each object can have its own set of views that can be customized with little programming effort. These views can be used to present information in ways that can enhance understanding of the data that is stored inside objects. A variety of visualization layouts are available to adapt and extend, such as tree views and graph structures. MReM leverages the extensive visualization support provided by GToolkit and applies it to visualize and manage requirements. Specifically, we implement visualizations for our pre-defined project requirements structure we established. This allows stakeholders to gain an overview of the project and the requirements progress at any point during development.

Linking capabilities. GToolkit is shipped with built-in support for annotating text inside documents. Words in a document can be adorned with special syntax to form links to objects within the environment, *e.g.*, classes or methods. We apply this functionality to the field of requirements engineering. By creating artifacts with textual descriptions that can be annotated, requirements can be linked to their corresponding source code. We use the annotation feature to implement a navigation system for tracing requirements to their implementation and vice versa.

4.2 MReM support for SDLC phases

Consider a software development company called “ESolutions”, that is responsible for creating a restaurant ordering system called “EWaiter”. In this company, there are multiple stakeholders responsible for each project: managers, team leaders, developers, and designers. The process of developing a new ordering system starts with requirements gathering activities, such as stakeholder meetings. During these meetings, all stakeholders brainstorm to come up with a list of requirements. Several artifact types are used to structure the requirements. Developers prefer elementary, actionable requirements, such as user stories and use cases. Managers deploy physical status boards for development monitoring. Finally, designers construct scenarios in text documents to document workflows and design choices.

The design process of the product entails further meetings with stakeholders, mainly designers,

developers, and managers. During these meetings, domain entities are derived from the artifacts for later development and put in a programming tool. Also, implementation points are assigned to each user story. These points are then used by the managers to structure sprints.

During development, daily scrum meetings are held at the status boards and other workflow artifacts. During the meetings, the stakeholders discuss any shortcomings, *e.g.*, user stories being behind schedule. Requirements are also traced from the artifacts to their implementation, *i.e.*, the classes in the programming environment that they reference. The work progress of the user stories is monitored by the managers, who propose strategies to improve workflow, *e.g.*, assigning more team members to certain requirements.

Finally, after deployment, maintenance activities have to be performed. For example, customers are consulted regularly about the performance of the product. Any suggestions for fixes or new features are documented in the artifacts, *e.g.*, modifications to the text documents containing scenarios. The story walls also have to be updated accordingly.

In this situation, one can see that the dispersion of artifacts presents some challenges. As the requirements change, they will have to be updated accordingly across all artifacts. This has to be done extremely carefully since inconsistencies across artifacts can be costly to identify and address. Furthermore, such dispersion of artifacts causes stakeholder groups to focus only on a subset of them. This can lead to loss of the "big picture" of a project, *e.g.*, developers not looking at the designed personas, and not understanding the final user's motives. Therefore, a central platform could be helpful for more efficient management and documentation of requirements.

The following section introduces the main features of MReM through its support for the four SDLC phases. For each phase, we discuss the supporting features of MReM. A more detailed description of the tool can be found in chapter 5.

4.2.1 Requirements phase

In the requirements phase, MReM supports project stakeholders by providing: (1) a pre-defined artifacts structure for documenting requirements, (2) containers for centralized storage of those artifacts, and (3) a Mind Map view to display and manage artifacts.

Custom artifacts structure. For documenting requirements, MReM offers a selection of artifacts modeled in a pre-defined hierarchical structure. As a starting point, we use *epics* as high-level artifacts, which are overarching objectives of the developed product. Their scope is large, and completing them requires extensive development effort, usually spread out over multiple Sprints. *Use cases* are used to break down the epics to present the principle functionalities of the system. Use cases define actors, who want to complete certain tasks. These tasks are broken down into concrete steps. Any preconditions are specified. Use cases are broken down into *user stories*, which are specific actions a user wants to achieve. Each user story includes a textual description from which possible domain entities to be implemented are extracted. User stories also contain implementation points and work status. The implementation points associated

with each user story reflect the amount of perceived effort needed to complete it. The work status, on the other hand, indicates if the story is currently being implemented. Our solution models three work states, namely: *Not started*, *In progress*, and *Complete*. Finally, every user story contains multiple *scenarios* that embody functional test cases for every possible outcome.

In the case of our sample company ESolutions, the managers responsible for the EWaiter system use this pre-defined structure to document their requirements. They can use the artifacts provided by MReM to construct their requirements base for EWaiter.

Containers. To manage the above artifacts, we use "requirement containers". In MReM, a requirement container is a class that provides users with an interface to display and manipulate project requirements. All project requirements are stored within the container, which simplifies their maintenance.

Before bootstrapping, a requirements engineer sets up a new requirement container for a project, see Figure 4.1. The container presents users with a description of the supported artifacts and associated views in the main window. Whenever a stakeholder navigates to the main container window, different views, such as "Mind Map" and "Story Wall", are available via the navigation bar above. This enables stakeholders to access and explore the requirements of a project in different formats. During the bootstrapping phase, project stakeholders gather and discuss the product requirements. Consequently, epics, use cases, user stories, and scenarios are derived. Once they are agreed upon, a non-technical stakeholder, such as a project manager, can use the Mind Map view in MReM to input those.

The containers help with the consistent storage of requirements; they are all in one place. Thus, the stakeholders of ESolutions always know where to look for the requirements of EWaiter.

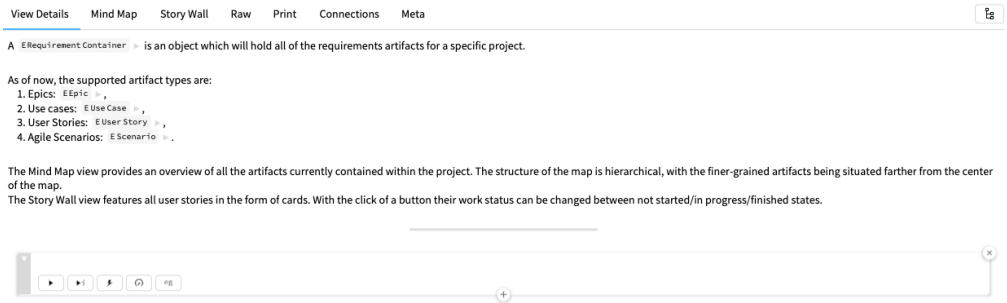

Figure 4.1: Initial View for an ERequirementContainer

Mind Map requirements management. To provide a clear overview of the project requirements in a container, we utilize a visualization inspired by mind maps. However, this approach can be easily extended to model other visual artifacts, *e.g.*, goal models. Our tool merely demonstrates the intended philosophy of our approach using a selection of artifacts. The principle idea is that different artifacts can be further modeled inside a single platform and maintained there.

The Mind Map view enables users to add requirements to the container, see Figure 4.2. This view is initially empty. At first, it provides the user with a button to add a new epic that opens an artifact addition menu, see Figure 4.3. These menus are custom for each artifact. They allow users to add custom information for each artifact, *e.g.*, assigning implementation points to a user story. Using such menus for manipulating the artifacts ensures their consistent data structure, *i.e.*, they contain a standardized set of attributes.

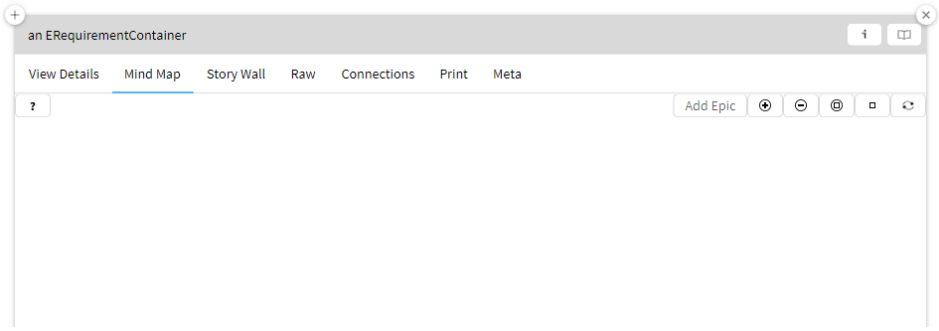

Figure 4.2: Initial Mind Map View

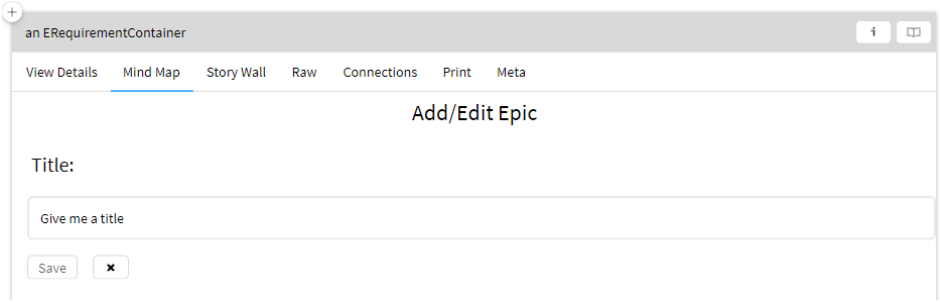

Figure 4.3: Addition Menu for an Epic

After creating an epic, the user can hover over the node representing it to explore a tooltip. This tooltip contains more information about a specific node, see Figure 4.4. It also provides buttons for editing and deleting the artifact, as well as adding a sub-artifact to it. For example, in the case of an epic, the user can create a use case associated with the epic. This enables stakeholders to identify requirements at a glance, and also expand the requirements structure all within one view. Besides, the constrained addition mechanism of adding child elements ensures that the hierarchical structure of the requirements is always maintained.

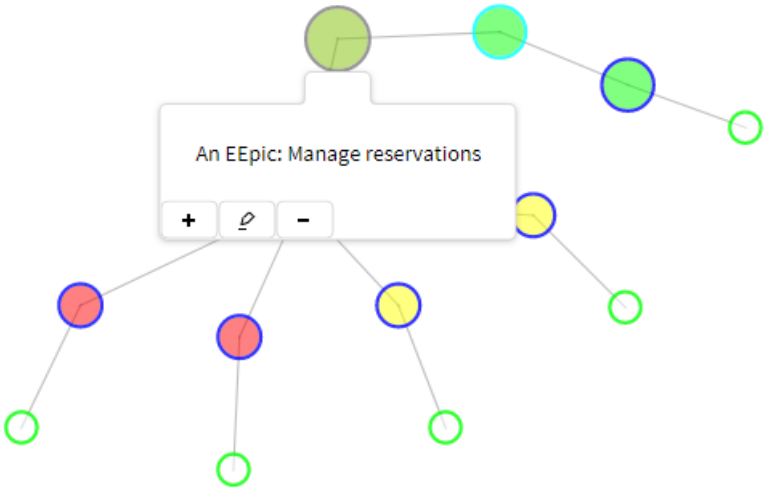

Figure 4.4: Tooltip element

The Mind Map view also provides a host of visualization dimensions, which we discuss in subsection 4.2.2.

Through the user-friendly interface of the Mind Map, any stakeholder of ESolutions can participate in the elicitation and maintenance of the requirements of EWaiter. For example, a manager can decide to add an epic which introduces a cashless payment system for EWaiter. Consequently, the team can discuss the epic and derive use cases to implement.

4.2.2 Design phase

In the design phase, MReM supports project stakeholders in the following ways: (1) we provide a visualization of the requirements in the Mind Map view to gain a quick overview, and (2) we use the same view for subsequent development and planning activities.

Gaining a quick overview of requirements. Once all project requirements have been inserted, the Mind Map view provides users with an overview of their structure, see Figure 4.5. This allows them to visualize the requirements effortlessly. Specifically, the Mind Map view incorporates the following visualization dimensions: (1) the border colors of nodes indicate what type of artifact they represent, *e.g.*, user stories incorporate blue borders; (2) the edges between the nodes visualize which artifacts are connected, *e.g.*, which use cases are associated with an epic; (3) the positions of nodes within the view illustrate their hierarchy, *i.e.*, the most high-level artifacts are nearest to the center of the view; (4) the node size indicates the number of implementation points assigned to the artifact; (5) finally, the background color indicates the work status of the artifact, *i.e.*, its level of completion. This dimension is discussed in more detail in subsection 4.2.3.

The above dimensions enable stakeholders to get an overview of a few vital aspects of the requirements environment, such as the hierarchical structure of the artifacts. The Mind Map view also allows users to reason about the existing requirements. For example, by using the tooltips, see Figure 4.4, the stakeholders of ESolutions can go over the structure of the payment system epic and its use cases. In case they reckon the epic is not fully covered, they can easily add another associated use case using the tooltip, *e.g.*, adding a use case to pay by phone.

Visual development planning. Once the refinement of the requirements structure is complete, stakeholders responsible for planning sprints can identify parts of the structure that are relevant for each sprint. Consequently, they can discuss with other stakeholders about possible corresponding design artifacts. For example, following discussions about which requirements to put into a sprint, the product owners decide to include some requirements artifacts. These were identified in the Mind Map view, *e.g.*, a use case with all of its associated child user stories and scenarios. Following the selection of these artifacts, stakeholders can brainstorm about relevant design artifacts to be created, *e.g.*, a sketch to illustrate interaction flow or domain models to illustrate object relations.

Additionally, in terms of planning the development processes, stakeholders can utilize the Mind Map view to get an overview of the implementation effort needed for each requirement. The node size of each element indicates how many points are assigned to it. In case the users require more detailed information about an element, they can click on the nodes and navigate to their implementation. In the case of EWaiter, product owners can use these features during planning. They can assign artifacts to sprints, and prioritize them according to how many resources (*i.e.*, developers and time) are available for each sprint. For example, if implementing the use case for the phone payment system is deemed cumbersome, the owner can assign more development teams to it.

4.2.3 Development and Testing phase

To support the development and testing phase, MReM provides: (1) a Story Wall view for the container that allows users to change work states of requirements, (2) the Mind Map view to visualize work progress, (3) a linking feature to connect requirements with their implementation, and (4) a simple interface for interactive modeling of the domain entities linked to each requirement.

Changing requirement work states. During development, the Story Wall view is used for changing the work states of user stories, see Figure 4.6. The story wall follows a structure of a status board that is divided into three columns. Each column corresponds to a work state: *Not started*, *In Progress*, and *Complete*. The columns are positioned in chronological order of the work states, from left to right. Each user story is displayed in the form of a story card that contains the most relevant information about it. During sprints, developers can change the work states of their assigned user stories within the board, and update it instantly. This notifies the managers that the requirements are in the process of being developed. For example, if the phone payment system of EWaiter contains two user stories, and the developers changed their states to be complete, the managers can then assign the developers to new requirements.

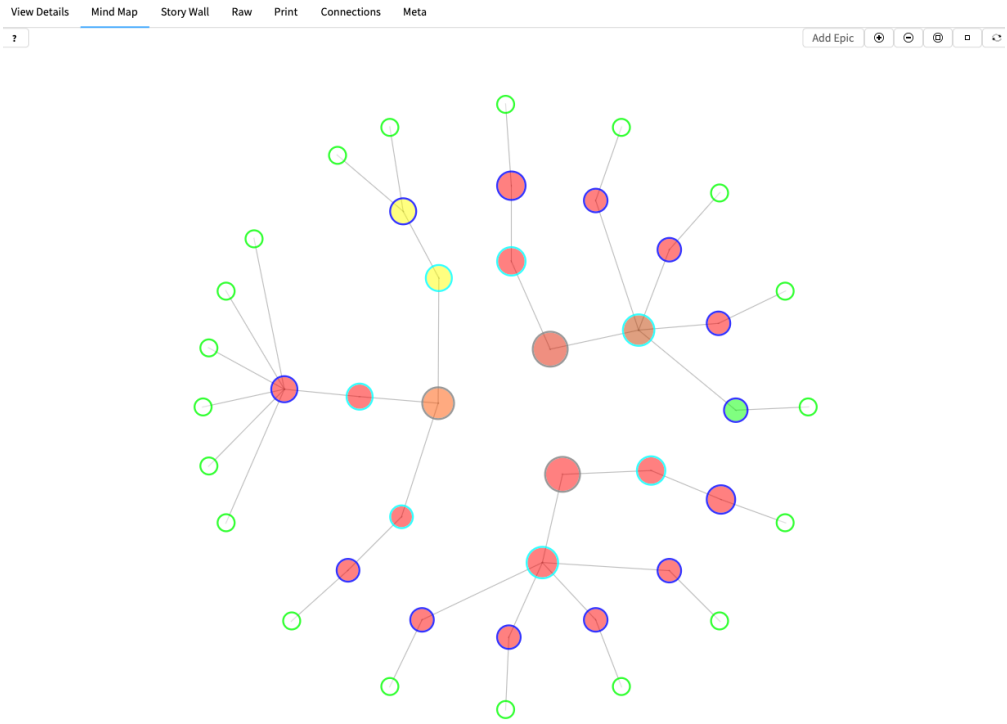

Figure 4.5: Sample Mind Map Visualization

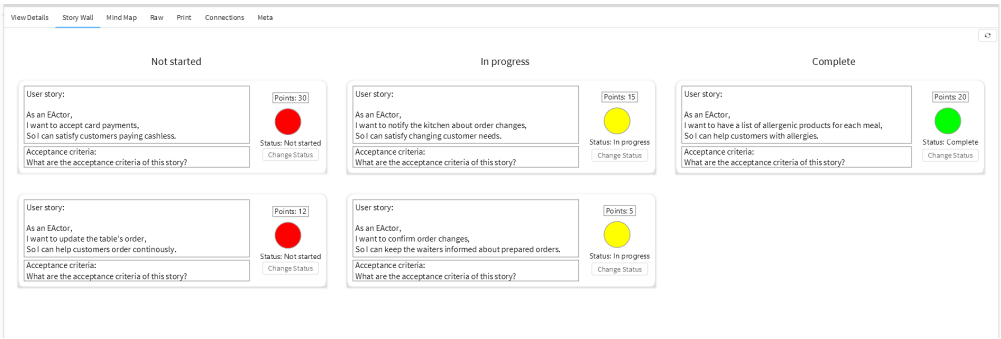

Figure 4.6: Sample Story Wall View

Visualizating work progress. To provide a visual way of monitoring the work state of requirements, we use the Mind Map view, see Figure 4.5. The color of the nodes indicates their work status, with the colors ranging from red (not started), through yellow (in progress), to green (complete). These colors visualize the work state of each requirement and allow stakeholders to instantly check the progression of the project. For example, during a sprint, the product owner of EWaiter can check the Mind Map to see if the use case for the phone payment system is in progress. Upon seeing the color of the nodes being too red, he can contact the appropriate developers and discuss the progress being made, or assign more developers to it.

This way, any stalls in productivity can be quickly addressed. Also, the view can be used to identify areas that are lacking in implementation, and plan future sprints accordingly.

Linking requirements with their implementation. At the level of user stories, we introduce a feature for linking requirements and their associated domain entities. Each user story has a description, parts of which can be linked with corresponding classes. For example, Figure 4.7 shows a user story description with added annotations. Classes that are not implemented are shown in red. During design meetings, user stories can be refined and appropriately annotated with classes to be implemented. For example, for the user stories associated with the cashless payment epic, one can expect entities such as "Phone", "Credit Card", or "Card Reader".

Figure 4.7: Sample Description View for a User Story

Additionally, looking at the Referenced Entities view for a user story, stakeholders can see which associated classes have already been created, see Figure 4.8. The view displays the list of references from the description. Each reference corresponds to a class.

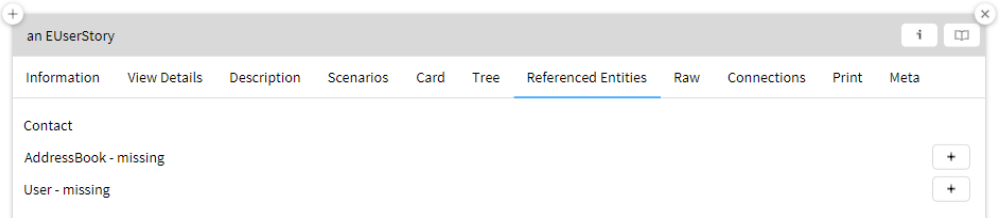

Figure 4.8: Sample Reference View for a User Story

Classes also have reference views, called "Requirement References", see Figure 4.9. These views enable users to see which requirements reference a class and subsequently navigate to those requirements. This makes the links between requirements and their implementation more apparent and explorable. The exploration of the classes and their associated requirements can put into perspective the importance of each class, *i.e.*, more referenced classes become a preferred target for quick completion.

For example, during development, the product owner of EWaiter can go from the cashless payment epic to a class that is referenced by it using the Referenced Entities view for the epic, see Figure 4.8. By clicking on a referenced class that has been created, namely the card reader class, the product owner is taken to its implementation. The Requirement References view for the class can be used to gain information about the related requirements, *e.g.*, how many use cases reference the card reader. This way, the product owner can discern how important the class is to implement, *e.g.*, if many epics reference a class, it needs to be implemented sooner, so that progress can be made in those epics. In the case of EWaiter, the card reader is referenced by three use cases, which makes it an important class to prioritize.

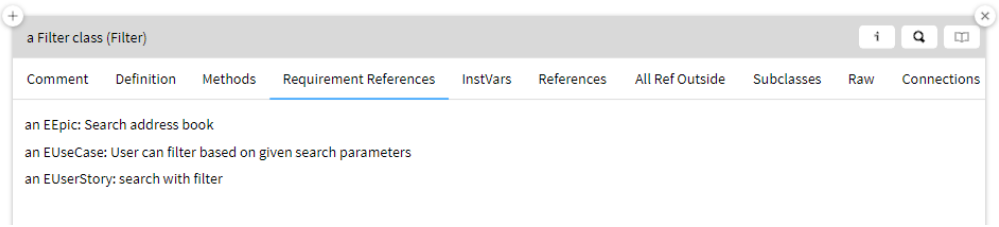

Figure 4.9: Sample Requirement Reference View for a Class

Interactive modeling of domain entities. Project managers can leverage the reference views described above to monitor development progress by observing which classes have been created, and how many requirements are in progress. Alternatively, by using a class addition menu within the view, see Figure 4.10, a stakeholder can quickly create a class corresponding to the referenced entity. The class addition menu is brought up by clicking on the button next to a missing entity in the Referenced Entities view. Instance variables and other class-related fields can be added within it. By using these menus, any stakeholder can set up an initial class structure for the developed product. This enables interactive and iterative modeling of the domain entities. For example, a team leader of EWaiter can model the card reader class and assign individual developers to implement that class.

Additionally, clicking on the reference item will take the user to the implementation of the class, so one can inspect it in detail. For example, the developers assigned to the card reader can navigate to the class created earlier and work on implementing it. This contributes to work monitoring and easy linking of the requirements with their implementation.

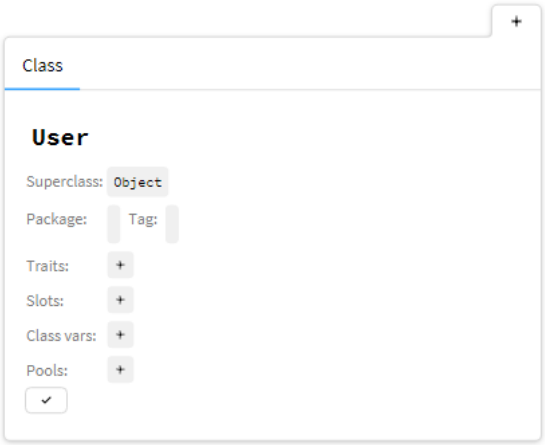

Figure 4.10: Class Addition Menu

4.2.4 Deployment and Maintenance phase

During the deployment and maintenance phase, we support stakeholders by keeping requirements easy to maintain and change, all within our tool. Specifically, MReM allows for: (1) flexible maintenance of the requirements through the Mind Map interface, and (2) monitoring of the code coverage of requirements through the Referenced Entities views.

Flexible requirements maintenance. As the project transitions to the deployment phase, project requirements are well-maintained within our platform. As discussed in the previous sections, the Mind Map view provides features for management and visualization of project requirements. For example, stakeholders can use the view to add new epics to implement new features or remove user stories which symbolize features that were deemed unpopular and not needed in a product. The view shows updated information about the work states of requirements, so whenever a new requirement needs to be implemented, stakeholders can be notified by looking at the updated Mind Map.

Software is frequently subject to change, even well past the initial deployment stage. Market trends, new technologies, and target consumer group shifts are among the reasons for these changes. In our tool, the requirements structure can be changed easily, and within the coding environment.

For instance, consider EWaiter, which has been deployed on the market. User reviews soon start coming in, and many people are complaining about the lack of a feature to reserve tables online. Imagine that within the requirements structure there is a use case that addresses reservations. By using the mind map interface, managers can add a new user story to the use case. This story can address users who want to book a table online (*e.g.*,, "As a user, I want to book a table online"). This story, upon being created, starts with the work state of Not started. Looking at the Mind Map view, the team leader can see a new requirement, which is red. Consequently, he assigns developers to add a feature for booking a table and notifying EWaiter of any new reservations. Thus, monitoring new changes and implementing them following the requirements is facilitated.

Monitoring the code coverage of requirements. The connections between artifacts and the source code and their Referenced Entities views enable easy monitoring of changes, even past deployment. For example, as new payment methods for EWaiter are added, their descriptions may reference new code entities, such as phones or RFID chips. The Referenced Entities views identify the classes that are not implemented and annotate them accordingly. Consequently, developers can use these views to get an overview of what needs to be implemented. Also, using the class addition menus, they can promptly set up these classes and afterward move to implement them in detail.

5 Implementation

In this chapter, we describe the two main building blocks of MReM, namely the requirement container and the linking system between requirements and code, from a technical point of view. Specifically, we provide detailed descriptions of the implemented features and supported workflows.

5.1 ERequirementContainer

The starting point for any new software project is the requirements container. It contains all of the requirements associated with the developed project. Figure 5.1 shows the supported views and objects of the container. The container stores requirements in the form of epics, use cases, user stories, and scenarios.

5.1.1 "View Details" View

The main window of ERequirementContainer hosts several views. The View Details view, see Figure 4.1, contains textual information about the supported artifacts, and briefly introduces the following views of the container. It helps new users to familiarize themselves with the artifact structure, as well as to help navigate to the other views. For example, non-technical stakeholders of ESolutions can use this view to gather information about how to navigate the container for EWaiter, and what artifact structure is present.

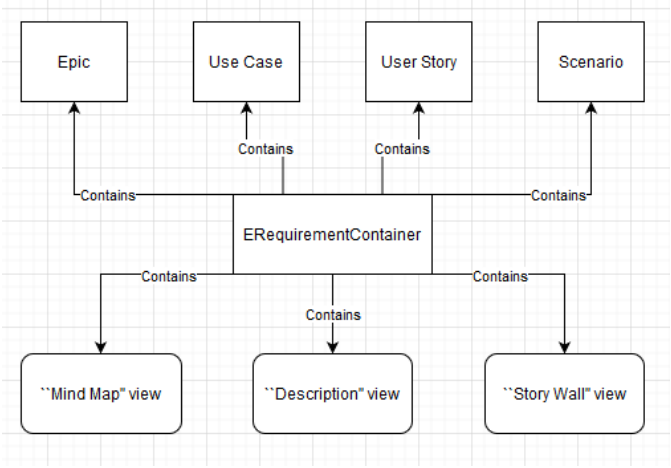

Figure 5.1: Requirements Container Diagram

The view is based on a GtDocument. The supported artifacts are annotated using the syntax outlined in Figure 5.8. This means that users can click on the annotated text and be taken to the implementation of the artifact.

5.1.2 "Mind Map" View

As a way to visualize requirements, the Mind Map view for the requirements container displays all of the associated artifacts in a graph with a cluster layout, see Figure 5.2. The diagram visualizes the functionalities of the view, see Figure 5.3.

The view displays all requirements artifacts from the container, *i.e.*, epics, use cases, user stories, and scenarios. Create, Read, Update and Delete (CRUD) functionality for all above artifacts is also present within the view. A button bar at the top of the pane allows the view to be manipulated. In addition, the "Help" button in the top left corner of the pane contains information about the visualization dimensions, *i.e.*, which dimensions visualize which aspects. Each artifact is represented by a node in the mind map.

The border color of each node corresponds to the type of artifact. The artifact types and their associated border colors are as follows:

- Epics — gray border;
- Use cases — teal border;
- User Stories — blue border;
- Scenarios — green border.

The edges connecting artifacts visualize their relationships. For example, if a use case has a connection with a user story, that story is a child of the use case. The relationships follow the hierarchy set by our tool,

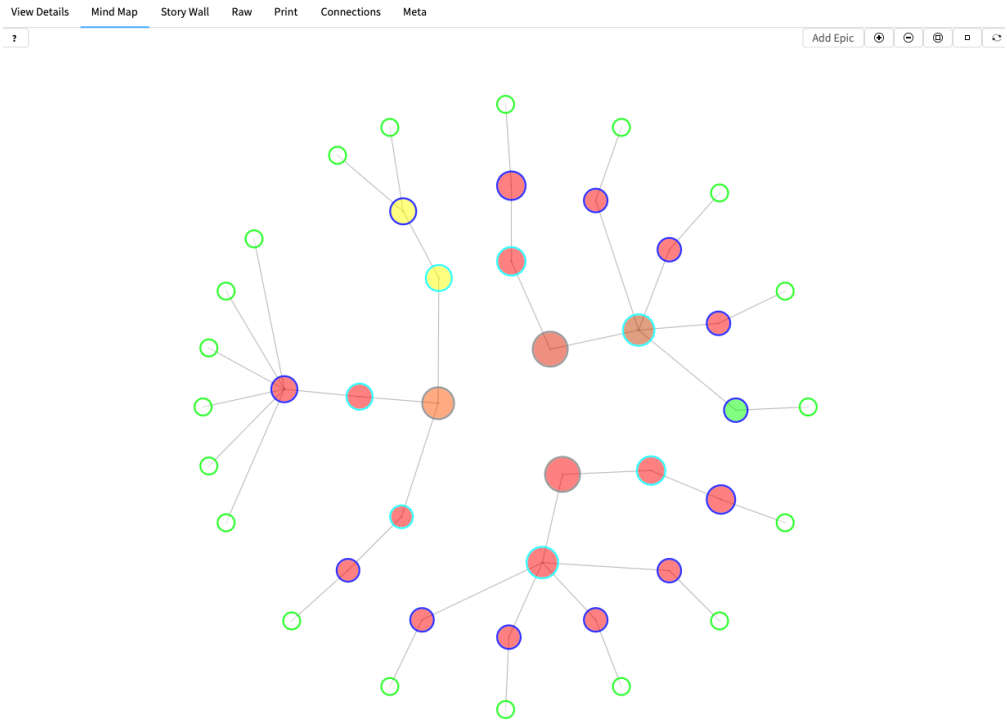

Figure 5.2: “Mind Map” View

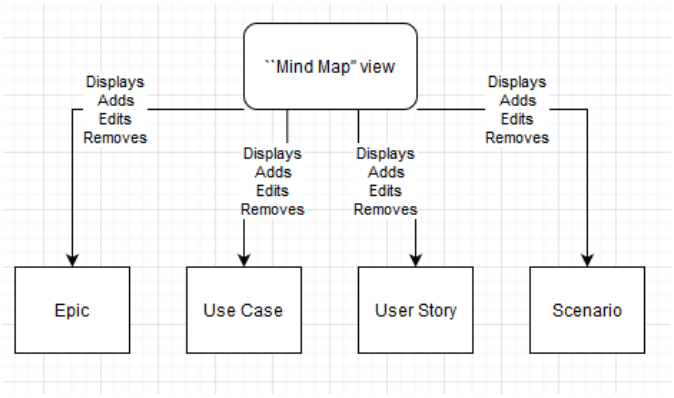

Figure 5.3: “Mind Map” View Diagram

i.e., epics, followed by use cases, then user stories, and finally, scenarios. The positions of the nodes also illustrate the hierarchy, *i.e.*, epics are at the center of the graph, and elements lower in the hierarchy are placed further away from the center.

The color of the node corresponds to its work status. Epics and use cases do not implement the notion of work status, therefore their default color is static, and does not symbolize anything. User stories have an associated work status; their color corresponds to their work status.

Once a user story is added in the container, its associated use case and epic inherit the status color of the user story. By default, a node has a red background color, which indicates that the implementation of its associated requirement has not been started. By changing the user story work state, the background color of its node and its associated parent nodes changes. When the state changes from "Not started" to "In progress", the background color changes to yellow. Once a user story is marked as complete, its color changes to green.

For parent elements, the node colors of the child elements are aggregated. Scenarios do not implement or inherit work status, so their background color is static.

The size of the node indicates the amount of implementation points assigned to it. Only user stories contain an associated field for implementation points. Use cases and epics aggregate the implementation points of their child elements and change their size accordingly. The node size is normalized to avoid cluttering of the view due to excessively large nodes. Scenarios do not implement or inherit implementation points. Their size is static.

Workflow. Upon creation of the requirements container for EWaiter, there are no artifacts associated with it. In our approach, an epic is the starting point for defining the requirements structure. To add an epic to the container, the user needs to navigate to the "Add Epic" button in the Mind Map view. Upon pressing it, a menu with input fields corresponding to the information of the epic appears, see Figure 4.3.

Upon inputting the data and clicking the "Save" button, an updated Mind Map view appears with the newly created epic, represented by a node.

Once the epic has been created, the user can hover over the node with the cursor to display an information tooltip, see Figure 4.4. This element contains basic information about the artifact, namely its type and title. Additionally, one can find a button bar on the bottom of the tooltip. Three buttons are present within:

- Add sub-element: Upon clicking this button, a menu for inputting data for the sub-element appears. The element's child artifact is determined by the hierarchy of our implemented requirements structure.
- Edit element: Enables the user to edit the selected element. The same menu appears as for adding the artifact of the given type, only with its values pre-filled in the input fields.
- Delete element: removes the selected element, along with all of its children, from the container.

By adding sub elements for epics, use cases, and user stories, the user can initialize all levels of the requirements structure, down to the most fine-grained artifact, *i.e.*, scenarios. These elements are automatically linked through establishing connections from a new artifact to its parent. This means that stakeholders can set up the requirements of EWaiter all within this view.

5.1.3 "Story Wall" view

The Story Wall view displays the user stories of a project inside three vertical columns, see Figure 5.4. The diagram for this view shows the workflows and relations, see Figure 5.5.

The view displays user stories in the form of story cards. The "Story Card" view allows the work status of the user story to be changed.

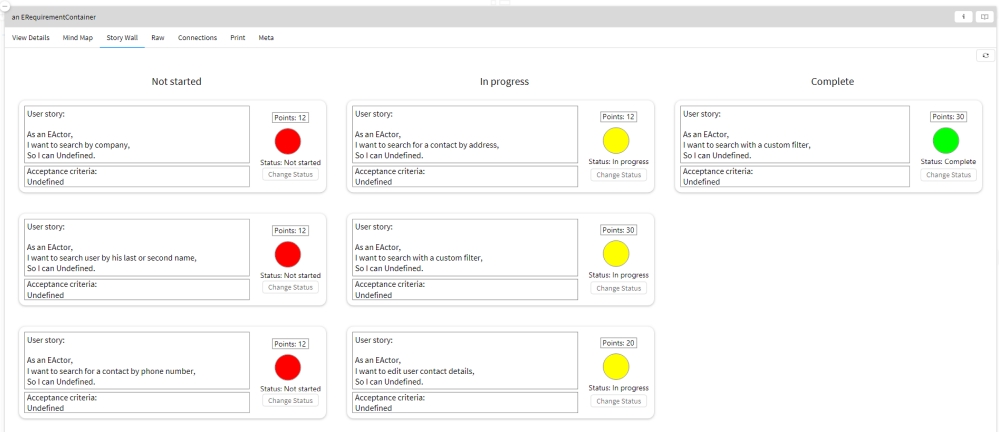

Figure 5.4: "Story Wall" View

The columns are positioned in chronological order of the work status, from left to right. Each user story is displayed in the form of a story card that contains the most relevant information about the user story, see Figure 5.6.

Workflow. To move the story cards through the columns, each story card contains a button in the lower right corner, with the label "Change status". Upon clicking it, the user story's status is changed, and the appropriate story card shifts to the next column. Subsequently, navigating back to the Mind Map view and clicking the "Refresh" button updates the view with the new work state values. The status changes are reflected in the colors of the nodes. For example, if a developer finished implementing a use case for card payments, he can indicate this by marking all of its associated user stories as complete. Consequently, the product owner can assign the developer to another requirement.

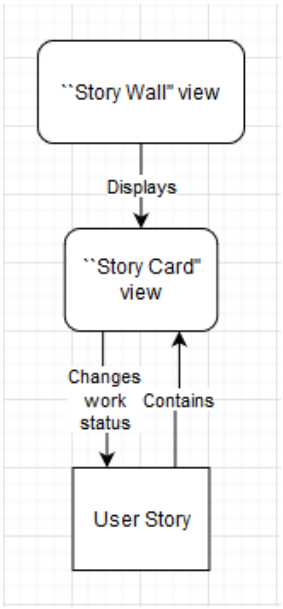

Figure 5.5: “Story Wall” View Diagram

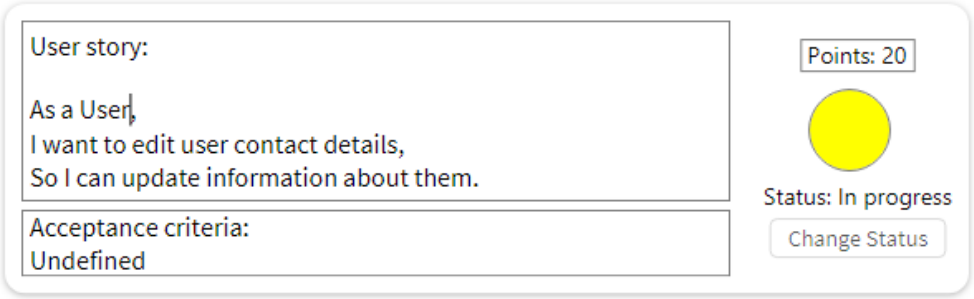

Figure 5.6: User Story Card

5.2 Linking Requirements with implementation

Towards tracing requirements to their associated source code, we present a two-way navigation system between requirements artifacts and their respective code references. For each user story, we implement a view of the domain entities referenced within its description.

5.2.1 “Description” View

Each user story contains a natural language description, which is imbued with references to domain entities (*i.e.*, classes). The description can be accessed through the Description view of each user story, see Figure 5.7. It displays the description in form of a GtDocument.

Figure 5.7: User Story "Description" View

To create references, we utilize the annotation functionalities of the Pillar markdown syntax, which is available in GToolkit documents. As an example, by taking a word from the description (*e.g.*, Contact) and enclosing it with the following syntax: $\${class:name=Contact}\$$, the word is turned into a reference within the document, see Figure 5.8.

Figure 5.8: Example of Description Annotation Syntax

Upon creation of the reference, the system searches its code base for a class with the corresponding name. If it is found, a button with a label containing the class name appears, and clicking on it takes the user to the implementation of the class. Conversely, if the class is not implemented, the corresponding text is highlighted in red, as in Figure 5.7.

Parent elements (*i.e.*, use cases and epics) aggregate the references from their associated user stories. Note that the reference extraction mechanism is implemented at the level of user stories. This means that scenarios do not contain references, and subsequently, the navigation system.

5.2.2 "Referenced Entities" View

For User stories, use cases, and epics, the list of referenced entities can be found in the view called "Referenced Entities", see Figure 5.9. If the class referenced by the list element is implemented, then only the name of the class is displayed. Otherwise, the name of the reference appears along with a caption notifying the user that it is missing from the coding environment, *i.e.*, it has not been implemented.

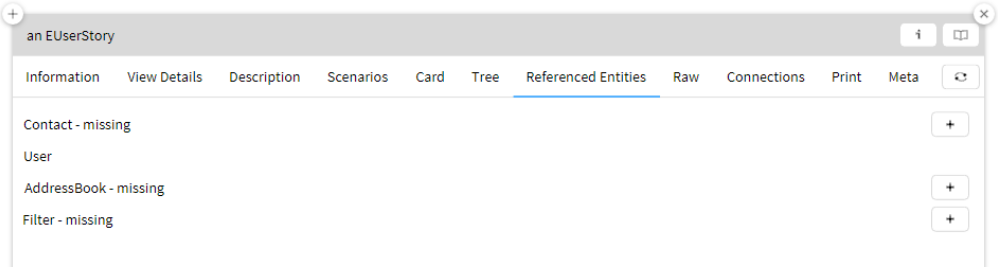

Figure 5.9: User Story "Referenced Entities" View

Additionally, a button is displayed on the right side of the unimplemented element, which gives the user the option to add a class with the corresponding name directly from the interface. Upon pressing it, a small window appears, with an interface for creating a new class in the environment, see Figure 5.10. The name field of the class to be added is pre-filled accordingly. Several options, such as declaring class and instance variables, are available.

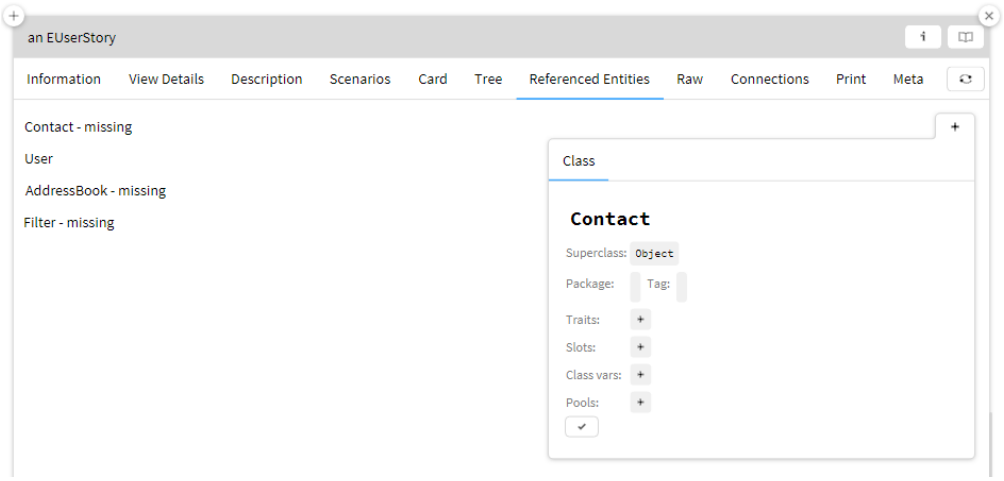

Figure 5.10: Class Addition Menu

Clicking the “Save” button on the bottom of the window creates the class and updates the view. The reference to the newly created class is now available in the view. This way, one can see which associated classes have been created, and explore them in more detail.

In the case of EWaiter, during stakeholder meetings user story descriptions can be added by the developers. They can initialize the class structure by creating references to them. By using the References Entities views, they arrive at a list of the classes which need to be implemented. Consequently, they can work on designing the interactions between these classes.

5.2.3 “Requirement References” View

For navigating from implementation to the requirements, navigating to any implemented class within the programming environment and selecting the Requirement References view displays a list of the requirements that reference it, see Figure 5.11. Each list element contains information about the requirement type, as well as its title. Clicking on an element takes the user to the corresponding artifact. This completes the two-way navigation system that we propose.

This can be used by project planners of EWaiter to check which classes are referenced most frequently by the requirements. They can then prioritize these classes for implementation during sprint planning.

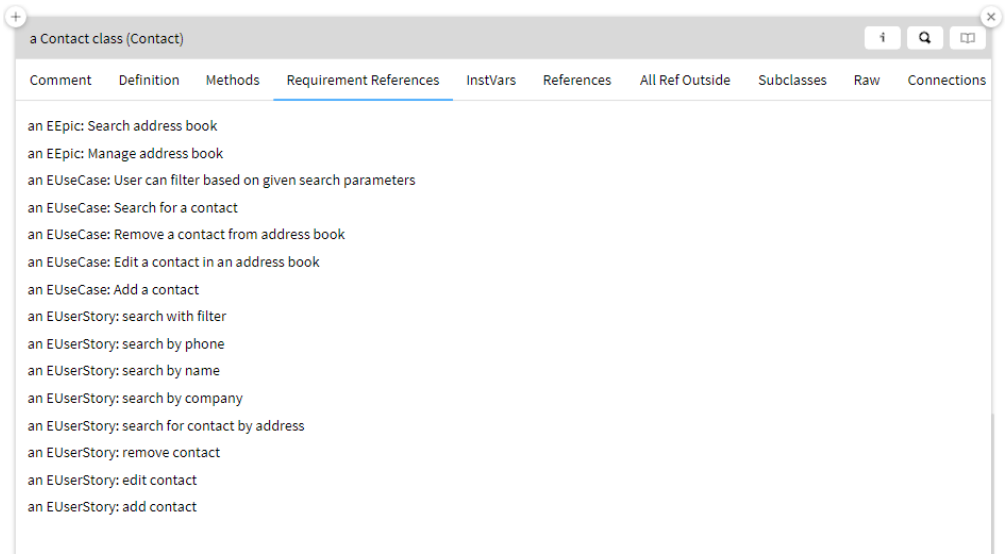

Figure 5.11: “Requirement References” View

6
Discussion

As outlined in chapter 2, most requirements engineering tools suffer from a few limitations. Specifically, management and modeling of requirements are weak points of these tools. Visualization is also not always utilized, *e.g.*, in the case of ProR. Our approach aims to address these issues.

Management. Requirements elicitation is the process that precedes their management. It stands to reason that proper elicitation has to be in place to set the project up for easier management. We posit that proper elicitation entails templating of artifacts to ensure their consistency. However, there is a noted lack of support for providing templates for elicitation [5]. Consequently, providing a consistent structure to document requirements and other related information needs improvement.

To address this gap, MReM provides templates for each artifact. For example, meetings can be held with MReM opened by developers or requirements engineers. The requirements can be directly inserted into the requirements container during the meetings. The modeled artifacts have their own pre-defined custom sets of values. These are set by the tool, so practitioners have to adhere to the set structures whenever they add or edit artifacts. This ensures a uniform structure of the requirements themselves, *i.e.*, there is no additional, unstructured information cluttering artifacts, improving their readability and manageability.

Another aspect of management is the storage of requirements. Earlier, we outlined how the dispersion of artifacts can cause problems in the areas of management. An example is ensuring changes are consistently

documented across all artifacts. In a dispersed requirements environment, some artifacts may be neglected in this process. For example, as a result of additional effort being required to search for the artifacts that reference a certain requirement, certain artifacts may get overlooked.

MReM addresses this issue by leveraging centralized artifact storage. The containers help keep all artifacts in one place and easily accessible. We reckon that this approach will result in fewer mistakes in the reflection of requirements changes.

Refinement of requirements is also important for ensuring their well-maintained structure. As the artifact structure becomes more complex, developers can try to discuss any unclear aspects of the requirements, *e.g.*, if a requirement is vaguely phrased. This can help clarify requirements for one of their intended recipients, the developers. Also, by utilizing the annotated user stories, developers can get a quick idea of the classes they need to implement.

An integral part of managing any software project is the planning of the development process. Resources need to be assigned to tasks, and deadlines need to be upheld. MReM provides features to streamline these activities.

Project planning can be done with the help of the Mind Map view. The size of the nodes indicates which requirements are perceived as more time-consuming to develop. Consequently, project owners can plan sprints and assign developers to requirements according to these estimates. By integrating workflow with the requirements that are to be developed, one can manage workflow more easily.

Various tools are available to developers during implementation activities. Issue trackers, such as JIRA,[1] focus on programming aspects such as bug fixes and release planning. However, these tools are disconnected from the development environment and present an additional interface to manage on top of operating an IDE. Our approach simplifies this by integrating workflow aspects and tracking inside a single tool. This keeps developers focused inside one tool, reducing navigation effort between windows and interfaces, which may be significantly different depending on the tools.

To streamline management during the development phase, MReM provides tools for monitoring workflow. A common way to manage workflow is using status boards. Tools such as Trello,[2] follow a pattern of information cards, which can be categorized according to their state. This resembles the Story Wall view in MReM. As with JIRA, such tools are not integrated into a single platform, which results in more navigation effort and having to learn a new interface. Our approach posits that keeping a single tool with a familiar interface and navigation system can lead to more efficient development and less learning overhead. Additionally, MReM's story cards are modeled directly in the programming environment and embody requirements artifacts directly. This simplifies navigation between visual representations and improves their integration with management processes.

[1] `https://www.atlassian.com/software/jira`
[2] `https://trello.com/`

As development advances, developers can change the work states of requirements using the Story Wall (Figure 4.6), notifying other stakeholders of current progress. The changes are also displayed in the Mind Map view, through node background colors. This lets stakeholders, such as team leaders, know how the work is progressing. Additionally, they can navigate to requirements, use their reference views (Figure 4.9), and navigate to the requirement's associated classes to get more detailed implementation information. Consequently, they can get a view of the work progress and sort out any issues, such as work delays, through discussion with developers. This link between requirements and their associated classes helps stakeholders quickly navigate to the relevant source code and inspect it. This alleviates traceability problems that stem from dispersed requirements environments.

Finally, thanks to linking requirements to their implementation in a live development environment, requirements can be more easily managed and maintained past development. Any artifact in the tool can be modified or deleted, and new ones can be added. This enables requirement engineers to respond to requirement changes quickly, and subsequently signal to developers the need to implement the changes in the code. Also, conversely to physical artifacts, such as physical story walls, all of our artifacts are integrated into one platform. Whereas a story wall can be difficult to maintain after deployment when linked with other, digital artifacts, our wall can be kept inside the project container. Any changes are easily performed within.

Modeling and Design. Our approach builds on the observation that varying combinations of artifacts are used to document requirements. As the number of types of artifacts increases, the likelihood of them being dispersed among media and tools also rises. MReM aims to contain the requirements environment within a single platform. To illustrate our approach, we modeled several requirements artifacts within MReM. However, any number and combination of artifacts can be implemented within. Each artifact corresponds to a code entity and can be molded to suit the needs of stakeholders. The modeling support of GToolkit lends itself to the flexible modeling of requirements. Depending on the preferences of companies, different combinations of artifacts can be modeled. This enables high flexibility in catering to stakeholder needs regarding the documentation of requirements.

Our approach also addresses the modeling of code. The reference views for requirements allow stakeholders to see which classes need to be in place for the requirement to be deemed completed. Additionally, the class addition menus for missing classes allow any stakeholder to set up a class structure for a project with little effort. This streamlines the transition from requirements to their implementation.

A selection of design tools, such as *Balsamiq*[3] or *Figma*[4] is available to RE practitioners. These tools focus on facilitating collaborative design. Specifically, significant emphasis is given to prototyping and interface design. However, neither of these tools provide functionalities for integration with artifacts outside the design field. For instance, domain model details are inaccessible in the above tools. This means that true integration into the development process is challenging. To address this issue, our approach demonstrates how design artifacts can be modeled and managed in a centralized way.

Towards designing the system itself, users can utilize the view to analyze different requirements, and create corresponding design artifacts, for example, sketches and domain models. By using a centralized approach, designs can be linked more easily with other artifacts. This way, design artifacts can be maintained in the environments of other stakeholders, such as developers, and bring them to focus on sketches or design concepts. This improves on the disconnected nature of the design tools mentioned before.

Visualization. Many stakeholders can benefit from having access to graphical representations of data. Visualizations help present information in a clean, effective way. We leverage visualizations in our tool to simplify RE.

Specifically, the Mind Map view provides a clear overview of a few essential aspects of a project. The structure and relations of the artifacts are visible in the Mind Map. The node size tells users which requirements are the most costly to develop. This can be used during sprint planning to structure the development process following the resource needs for each requirement.

[3] `https://balsamiq.com/`
[4] `https://www.figma.com/`

7

Conclusion and future work

Requirements engineering practices are determined in large part by the artifacts teams use to communicate and manage requirements. Numerous artifacts are available to stakeholders, with their characteristics and formats. As the number of used artifacts increases, it is challenging to maintain and ensure consistency of requirements across them. We speculate that this stems from the dispersion of artifacts among media and tools.

Our analysis of requirements artifacts led us to the conclusion that management of requirements can pose various challenges when they are distributed. Specifically, reflection of requirements changes across all artifacts requires additional effort. Additionally, different stakeholders may not pay attention to certain artifacts if they are not maintained in their field of focus, *e.g.*, design personas being kept separate from a programmer's toolkit. Making requirements clear to developers is also an important objective. Finally, a lack of structure prevents the stakeholders to gain an overview of the various dependencies between artifacts.

Following these observations, we outlined an approach of requirements management within a single platform. Our design philosophy lays the foundation of centralized requirements modeling and management tool. MReM, demonstrates how such an approach could look like in practice. Through modeling artifacts and utilizing different views for them, stakeholders can gain a better understanding of the requirements structure, and manage them more efficiently.

We believe that our tool, MReM, provides a way to manage and model requirements efficiently. However, improvements can still be made to support a wider range of artifacts and stakeholders. Depending on the development setting and stakeholder preferences, different combinations of artifacts can be modeled with little programming effort. Furthermore, a vast array of visualizations is programmable to model additional artifacts. For instance, designers could be supported by including design artifacts in the tool, *e.g.*, personas and scenarios.

Multiple data formats are used to document requirements. Many of them are established in the requirements industry. Formats such as ReqIF and BPMN are commonly used as an exchange medium between different companies. Their rigid, universal structure allows them to be manipulated by different RE tools. MReM could be improved to support these formats as well. For example, the formatted files could be parsed by the tool to create whole requirements structures from scratch.

The current features are intended to be a stepping stone towards creating a more extensive, custom RE platform. If the guiding principle of modeling artifacts inside a single platform is kept in mind, practitioners can extend the functionality of such tools almost endlessly. We hope to stimulate discussion about optimal ways to display and manage requirements, and to put those ideas into practice in integrated platforms such as ours.

Bibliography

[1] U. Abelein and B. Paech. A proposal for enhancing user-developer communication in large IT projects. In *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE)*, pages 1–3, 2012.

[2] J. M. Bass. Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology*, 75:1 – 16, 2016.

[3] J. K. Blomkvist, J. Persson, and J. Åberg. Communication through boundary objects in distributed agile teams. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1875–1884, 2015.

[4] J. Buchan. An empirical cognitive model of the development of shared understanding of requirements. In *Requirements Engineering*, pages 165–179. Springer, 2014.

[5] J. M. Carrillo de Gea, J. Nicolás, J. L. F. Alemán, A. Toval, C. Ebert, and A. Vizcaíno. Requirements engineering tools. *IEEE Software*, 28(4):86–91, 2011.

[6] N. R. Darwish and S. Megahed. Requirements engineering in Scrum framework. *International Journal of Computer Applications*, 149(8):24–29, 2016.

[7] A. Garcia, T. S. da Silva, and M. Selbach Silveira. Artifacts for agile user-centered design: a systematic mapping. *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

[8] M. N. A. Khan, M. Khalid, and S. ul Haq. Review of requirements management issues in software development. *International Journal of Modern Education and Computer Science*, 5(1):21, 2013.

[9] O. Liskin. How artifacts support and impede requirements communication. In S. A. Fricker and K. Schneider, editors, *Requirements Engineering: Foundation for Software Quality*, pages 132–147, Cham, 2015. Springer International Publishing.

[10] O. Liskin, K. Schneider, F. Fagerholm, and J. Münch. Understanding the role of requirements artifacts in Kanban. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 56–63, 2014.

[11] G. S. Matharu, A. Mishra, H. Singh, and P. Upadhyay. Empirical study of agile software development methodologies: A comparative analysis. *SIGSOFT Softw. Eng. Notes*, 40(1):1–6, Feb. 2015.

[12] N. Patkar. An emerging overview of requirements engineering tools. unpublished, 2020.

[13] M. Plachkinova, K. Pfeffers, and G. Mood. Communication artifacts for requirements engineering. *Donnellan B., Helfert M., Kenneally J., VanderMeer D., Rothenberger M., Winter R. (eds) New Horizons in Design Science: Broadening the Research Agenda. DESRIST 2015. Lecture Notes in Computer Science, vol 9073. Springer, Cham*, 2015.

[14] C. Rolland. Modeling the evolution of artifacts. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 216–219. IEEE, 1994.

[15] N. B. Ruparelia. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, 35(3):8–13, May 2010.

[16] E.-M. Schön, J. Thomaschewski, and M. J. Escalona. Agile requirements engineering: A systematic literature review. *Computer Standards & Interfaces*, 49:79–91, 2017.

[17] E.-M. Schön, J. Thomaschewski, and M. J. Escalona. Identifying agile requirements engineering patterns in industry. Association for Computing Machinery, 2017.

[18] S. Sharma, D. Sarkar, and D. Gupta. Agile processes and methodologies: A conceptual study. *International journal on computer science and Engineering*, 4(5):892, 2012.

[19] G. E. Stark, P. Oman, A. Skillicorn, and A. Ameele. An examination of the effects of requirements changes on software maintenance releases. *Journal of Software Maintenance: Research and Practice*, 11(5):293–309, 1999.

[20] S. Winkler. Information flow between requirement artifacts. results of an empirical study. In P. Sawyer, B. Paech, and P. Heymans, editors, *Requirements Engineering: Foundation for Software Quality*, pages 232–246, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

A

Appendix A

A.1 Identified artifacts

Table A.1: Classification of the identified requirements artifacts

Artifact	Physical?	Digital?	Format
User stories	·	✓	Textual
Story cards	✓	·	Mixed
Storyboards	✓	✓	Graphical
Kanban board	✓	✓	Mixed
User journey	·	✓	Mixed
Pin board	✓	✓	Mixed
Index cards	✓	·	Mixed
Release plan	·	✓	Mixed
Status board	✓	✓	Mixed
Story estimates	·	✓	Textual
Post-it notes	✓	·	Mixed
Design concept	·	✓	Graphical
Prototype	✓	✓	Mixed
Mockups	·	✓	Mixed
Wireframes	·	✓	Graphical
Interaction scenarios	·	·	Mixed
Pictures	✓	✓	Graphical
Videos	·	✓	Graphical
Sketches	✓	✓	Graphical
Screenshots	✓	✓	Graphical
MVP	✓	✓	Mixed
UML diagrams	·	✓	Mixed
EMF models	·	✓	Mixed
Domain models	·	✓	Graphical
Process model	·	✓	Mixed
User interface model	·	✓	Mixed
System model	·	✓	Mixed
Data model	·	✓	Mixed
Goal models	·	✓	Graphical
Role model	·	✓	Graphical
User model	·	✓	Mixed
Ontological models	·	✓	Mixed
Business Process Models	·	✓	Graphical
Task model	·	✓	Graphical
Node and edge models	·	✓	Graphical
Test specifications	·	✓	Textual
Integration tests	·	✓	Textual
Regression tests	·	✓	Textual
Unit tests	·	✓	Textual
Test plan	·	✓	Textual
Mind maps	·	✓	Graphical
Roadmap	✓	✓	Mixed
Impact map	·	✓	Graphical
Effect maps	·	✓	Graphical
Story map	✓	✓	Mixed
Vision	·	✓	Mixed
Tasks	·	✓	Textual
Scenarios	·	✓	Mixed
Personas	·	✓	Mixed
Issues	·	✓	Textual
Product backlog	✓	✓	Mixed
Burn down chart	✓	✓	Graphical
Epic	·	✓	Textual
Features	·	✓	Textual
Use-cases	·	✓	Mixed
Use-case description	·	✓	Textual
User wish list	·	✓	Textual
Feature diagram	·	✓	Graphical
Tags	·	✓	Textual
Business rules	·	✓	Textual
DSL	·	✓	Textual
Excel reports	·	✓	Mixed

Table A.2: Artifacts contained within other artifacts

Artifact / Contains	User stories	Story cards	Index cards	Release plan	Story estimates	Post-it notes	Sketches	Goal model	Test specifications	Tasks	Scenarios	Personas	Epic	Features	Use-cases	Use-case description	Excel reports
User stories	·	·	·	·	✓	·	·	·	·	✓	·	·	·	·	·	·	·
Story cards	·	·	·	·	·	·	·	·	·	✓	·	·	·	·	·	·	·
Storyboards	·	·	·	·	·	·	✓	·	·	·	·	·	·	·	·	·	·
Kanban board	·	✓	✓	·	·	✓	·	·	·	·	·	·	·	·	·	·	·
Pin board	·	·	✓	·	·	✓	·	·	·	·	·	·	·	·	·	·	·
Release plan	·	·	·	·	·	·	·	·	·	✓	·	·	·	✓	·	·	·
Status board	·	✓	✓	·	·	✓	·	·	·	·	·	·	·	·	·	·	·
Interaction scenarios	·	·	·	·	·	·	·	·	·	·	·	✓	·	·	·	·	·
UML diagrams	·	·	·	·	·	·	·	·	·	·	·	·	·	·	✓	·	·
Process models	✓	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
User interface model	·	·	·	·	·	·	·	·	✓	·	·	·	·	·	·	·	·
Goal models	·	·	·	·	·	·	·	·	·	·	✓	✓	·	·	·	·	·
Test specifications	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	✓
Mind maps	·	·	·	✓	·	·	·	·	·	·	·	·	·	·	·	·	·
Impact map	·	·	·	·	·	·	·	✓	·	·	·	·	·	·	·	·	·
Effect maps	·	·	·	·	·	·	·	✓	·	·	·	·	·	·	·	·	·
Story map	·	✓	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
Vision	·	·	·	·	·	·	✓	·	·	·	·	·	·	·	·	·	·
Scenarios	·	·	·	·	·	·	·	·	·	·	·	✓	·	·	·	·	·
Product backlog	✓	·	·	·	·	·	·	·	·	✓	·	·	✓	·	·	·	·
Use-cases	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	✓	·
Feature diagram	·	·	·	·	·	·	·	·	·	·	·	·	·	✓	·	·	·

Declaration of consent

on the basis of Article 30 of the RSL Phil.-nat. 18

Name/First Name:

Registration Number:

Study program:

Bachelor ☐ Master ☐ Dissertation ☐

Title of the thesis:

Supervisor:

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 36 paragraph 1 litera r of the University Act of 5 September, 1996 is authorized to revoke the title awarded on the basis of this thesis.
For the purposes of evaluation and verification of compliance with the declaration of originality and the regulations governing plagiarism, I hereby grant the University of Bern the right to process my personal data and to perform the acts of use this requires, in particular, to reproduce the written thesis and to store it permanently in a database, and to use said database, or to make said database available, to enable comparison with future theses submitted by others.

Place/Date

Signature