Moldable Requirements

Nitish Patkar
Software Composition Group
University of Bern
Switzerland
Email: http://scg.unibe.ch/staff

***Abstract*—Separate tools are employed to carry out individual requirements engineering (RE) activities. The lack of integration among these tools scatters the domain knowledge, making collaboration between technical and non-technical stakeholders difficult, and management of requirements a tedious task. In this Ph.D. research proposal, we argue that an integrated development environment (IDE) should support various RE activities. For that, distinct stakeholders must be able to effortlessly create and manage requirements as first-class citizens within an IDE.**

With "moldable requirements," developers create custom hierarchies of requirements and build tailored interfaces that enable other stakeholders to create requirements and navigate between them. Similarly, they create custom representations of requirements and involved domain objects to reflect various levels of detail. Such custom and domain-specific representations assist non-technical stakeholders in accomplishing their distinguished RE related tasks. The custom interfaces make the IDE usable for non-technical stakeholders and help to preserve requirements in one place, closer to the implementation.

I. Introduction

Requirements engineering (RE) is a phase of the software development lifecycle (SDLC), where user requirements are collected and transformed into system ones to be eventually implemented [1], [2]. Several techniques and tools have sought to automate the RE process and enable collaboration among stakeholders. There is a variety of proprietary and open-source tools proposed in the industry and academia to support distinct RE activities. Analysis of these tools, among other things, revealed that: (1) most of them are intended to support a particular RE activity and have a specific target audience, and (2) they offer limited support for IDEs, often through plugins. Due to such specialization of the tools for one or few RE activities and distinct target audiences, requirements scatter through many documents and are maintained in numerous formats. Therefore, facilitating collaboration among stakeholders to iteratively build a product vision also means making the employed tools to interact with each other.

Let us consider building an address book application that allows users to create and add contacts to an address book. Although very small, this example will allow us to discuss the underlying ideas nicely. Requirements for this application could be maintained in different formats and in several tools, from high-level ones, such as epics [3] in a platform like Jira [4], down to concrete scenarios [5] in a separate tool like Cucumber [6]. In addition to specifying requirements, domain experts need to verify whether developers have accurately implemented the requirements. They achieve it either by running tests or by interacting with the running application using a user interface (UI). Both verification approaches have limitations, as testing merely asserts how much functionality is actually working and does not warrant that the application behavior is modeled correctly. Likewise, building a UI costs time and effort [7].

As a solution, researchers have proposed to specify requirements directly in an IDE. However, this research field is less explored compared to proposing new tools and techniques and exhibits several challenges that need attention: (1) understanding different requirements formats and their corresponding characteristics to determine the usefulness for distinct stakeholders, and (2) determining efficient navigation strategies between requirements to make them accessible to all stakeholders. We argue that requirements should be specified, maintained, and implemented as first-class citizens within an IDE to truly justify the integrated nature of IDEs. To support the RE process and to engage non-technical stakeholders, the IDE must enable effortless creation and maintenance of requirements. For that, developers must build requirements hierarchies, *i.e.,* model high-level formats, such as epics, and other more specific formats, such as user stories [3] as first-class entities. Additionally, they must also provide appropriate interfaces to iteratively create, manage, and link the corresponding requirements to the implementation [8].

With moldable requirements, we suggest that requirements hierarchies, as well as their representations, must be adapted (*i.e., molded*) to suit the application domain and project needs. Developers first create custom requirements hierarchies. A model hierarchy of requirements to implement the address book example might involve creating epics, use cases, user stories, and scenarios as first-class entities in an IDE. Next, developers build interfaces, such as graphical ones, that enable other stakeholders to create, access, and navigate the corresponding requirements. For the address book application, stakeholders can use buttons and forms to create and save epics, *etc.* Likewise, they can use graph structures to navigate from epics to associated user stories. Developers also craft domain-specific representations for the involved domain entities in the requirements. Such domain-specific representations will enable non-technical stakeholders to inspect a *contact* with a contact card representation built by the developers. The same approach allows us to represent design models, such as the use case diagram, as just another representation of a particular instance of a use case. To realize this vision, we pose the following hypothesis and aim to answer the following research question.

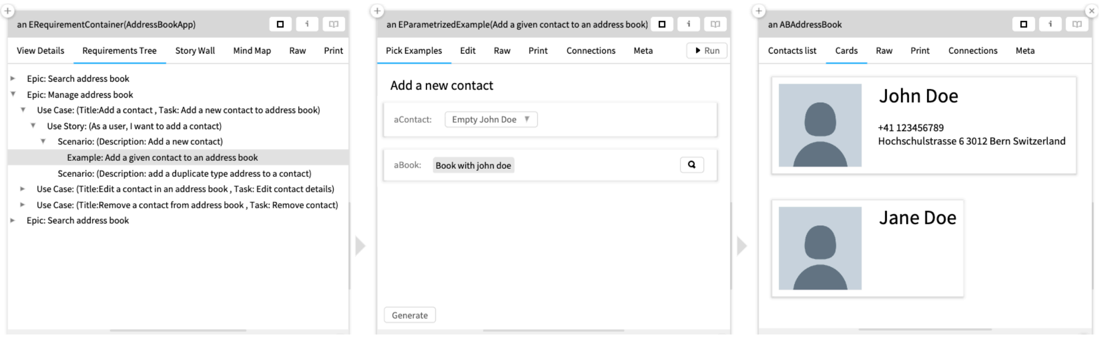
Hypothesis. An IDE could be used to support various RE activities given a mechanism to build appropriate interaction interfaces for both technical and non-technical stakeholders.

Research question. What features must an IDE exhibit and what infrastructure needs to be built to enable distinct stakeholders to actively participate in the iterative RE process?

II. State-of-the art

Researchers, through approaches such as user-centered design [9], behavior-driven development (BDD) [10], and domain modeling [11], have attempted to: (1) facilitate agile collaboration among stakeholders by proposing specification formats that everybody in a team easily understands, and (2) enable requirements specification in an IDE, mostly through developing plugins. User stories, originating from user-centered design, define the application behavior from the end-user perspective and are maintained in one of the tools. Often these tools are isolated from the development environment, which leads to traceability issues. Some of the proposed tools offer IDE integration; however, the characteristics of the available integration are not studied in the existing research [12]. With BDD, non-technical stakeholders specify application behavior in a constrained natural language format. The behavior is tested by linking the specification to the implementation through automatically generated test cases [10]. The success of BDD depends heavily on the employed tools [13], [14]. Our analysis revealed that many popular BDD tools already support BDD workflow in IDEs through plugins. However, these plugins offer limited capabilities for specifying behavior as they solely focus on textual specification formats. Similarly, they provide limited opportunities for non-technical stakeholders to verify the details about the specified behavior as the tools mostly output a test status.

In contrast to natural language specifications, requirements modeling approaches suggest using formal notations. There is an extensive support for requirements and domain modeling in IDEs [15], [16]. Model-driven approaches, such as model-driven development (MDD) and model-driven engineering (MDE), encourage expressing the application domain using concepts that are independent of



Epic: Search address book
Epic: Manage address book
Use Case: (Title:Add a contact , Task: Add a new contact to address book)
Use Story: (As a user, I want to add a contact)
Scenario: (Description: Add a new contact)
Example: Add a given contact to an address book
Scenario: (Description: add a duplicate type address to a contact)
Use Case: (Title:Edit a contact in an address book , Task: Edit contact details)
Use Case: (Title:Remove a contact from address book , Task: Remove contact)
Epic: Search address book

Add a new contact
aContact: Empty John Doe
aBook: Book with john doe

John Doe
+41 123456789
Hochschulstrasse 6 3012 Bern Switzerland

Jane Doe

Fig. 1: Moldable requirements approach

underlying implementation technology, which facilitates communication between team members [17]. However, such approaches have received limited appreciation in practice due to extensive required training and laborious efforts for specifying detailed models [18].

III. Moldable requirements

The term "moldable requirements" extends the concept of "moldable development" to refer to an environment that enables distinct stakeholders to specify requirements at different levels of detail and verify with representations specific to a particular application domain [19]. Such a moldable environment supports the creation of custom (i) requirements hierarchies, and (ii) requirements representations. Custom hierarchies make requirements navigable from higher-level ones down to the involved domain objects. Custom representations make the relationships between requirements explicit and aid domain experts in inspecting the modeled domain entities.

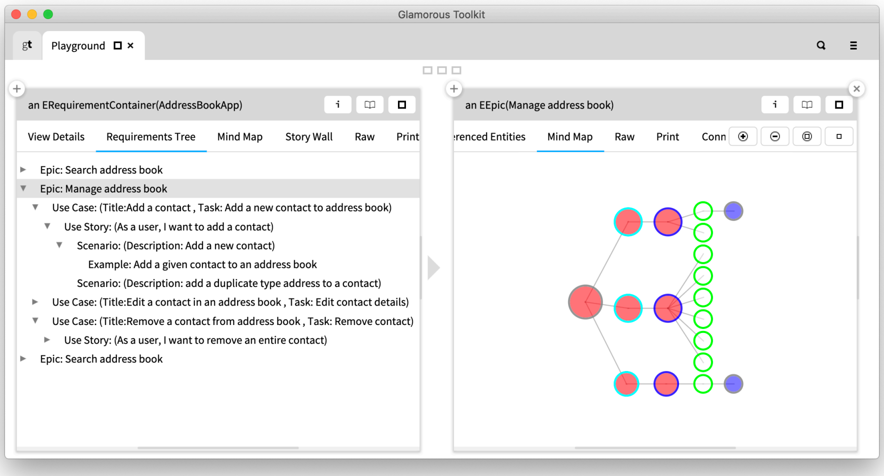
Figure 1 shows a prototype implementation of a project specific requirements hierarchy for the toy address book example. Users navigate between two windows. The second window enables domain experts to edit a particular scenario. Here, a user edits a scenario of adding a contact to an address book by selecting an *address book* and a *contact* and inspects the resulting address book. The third window shows the result of the changes (*i.e.,* the resulting address book). Such domain-specific representation enables non-technical stakeholders to inspect the details of the domain objects more efficiently compared to a test status. This hierarchy and the corresponding tree representation is custom created by developers. The tree representation enables other stakeholders to effortlessly navigate between requirements, as well as to gain a general overview of the existing ones. Figure 2 highlights the idea of custom representations. The tree view from Figure 1 for the requirements hierarchy is represented with another interactive visualization. This graphical interface facilitates the creation and navigation between requirements. In other words, it enables non-technical stakeholders to create and access requirements without any programming overhead. To validate such a generic approach with a controlled experiment or a field study is cumbersome. Every case has different conditions depending on the task at hand and people performing the RE activities for a specific domain. Therefore, instead of relying on experiments or empirical studies, one can validate the approach by demonstrating how easy it is to adapt it to several domains.

IV. Contributions

The following projects lead us to accomplish the discussed vision for "moldable requirements" approach.

A. Current contributions

RE tools survey. The study presents a comprehensive overview of 112 RE tools proposed at the top software engineering (SE) venues during the past five years. We reviewed a total of 203 publications and identified 112 tools that support one of the



```
▸ Epic: Search address book
▾ Epic: Manage address book
    ▾ Use Case: (Title:Add a contact , Task: Add a new contact to address book)
        ▾ Use Story: (As a user, I want to add a contact)
            ▾ Scenario: (Description: Add a new contact)
                Example: Add a given contact to an address book
              Scenario: (Description: add a duplicate type address to a contact)
    ▸ Use Case: (Title:Edit a contact in an address book , Task: Edit contact details)
    ▾ Use Case: (Title:Remove a contact from address book , Task: Remove contact)
        ▸ Use Story: (As a user, I want to remove an entire contact)
▸ Epic: Search address book
```

Fig. 2: Moldable requirements- multiple views

several RE activities. The findings indicate a lack of tools that support multiple RE activities. Likewise, activities, such as requirements management, are largely neglected.

Moldable scenarios. The study presents a review of 14 popular BDD tools, reports their characteristics regarding the support for input and output formats and interfaces for distinct stakeholders in an IDE. We observed that the existing BDD tools are vastly developer-oriented in the functionalities they offer. As a result, they poorly engage other stakeholders in the BDD process. Commonly used BDD tools facilitate linking textual specifications to the corresponding implementation through the glue code. For behavior verification these tools only display test status as an output. We present the "moldable scenarios" approach and an advanced prototype implementation that demonstrates effective integration of the BDD process into an IDE. Non-technical stakeholders can leverage graphical widgets to build complex domain objects and use those objects to compose behavior tests. Subsequently, they run the tests and inspect the output that is presented to them with custom representations.

Moldable artifacts. There is a lack of research that studies the characteristics of the numerous requirements and software artifacts to understand their suitability for distinct stakeholders and analyze their flow within the SDLC. We present a comprehensive overview of 62 artifacts and discuss their characteristics, such as format, nature, phases of origin, and usage. We observed that a lot of artifacts are produced during the requirements gathering and design phases, but most of them are used during the development and maintenance phases. To simplify artifacts management across isolated tools, we present an advanced prototype implementation of the "moldable artifacts" approach, wherein we model a selection of artifacts in an IDE as first-class entities.

B. Planned contributions

Survey of glue code properties in BDD tools. There is little evidence in the existing literature how much glue code is auto-generated by the BDD tools and how much must be manually written to connect behavior specification and respective test code. Researchers have recently published two datasets that contain open-source projects that use BDD tools [13], [14]. We plan to conduct a study that takes a closer look at the characteristics of the glue code provided by the BDD tools in these projects.

Living user stories. There are several tools proposed for agile project management that specifically facilitate user story creation, editing, and management. Existing studies have attempted to identify functional requirements for such tools and consequently classified them to see if they fulfill these requirements. However, none of the studies tried to analyse the support of such tools within IDEs. In this project, we will review a selection of user story management tools to uncover their limitations regarding support in IDEs. A prototype implementation will demonstrate a model of user story wall to manage user stories in an IDE.

Moldable graphical actor modeling. A lot of graphical modeling tools are proposed and used in practice. In this project, we will study the characteristics of a selection of graphical modeling tools. In particular, we will take a comprehensive look at their support in an IDE and bi-directional change propagation mechanism between model code and source code they provide. We will present an advanced prototype implementation that enables non-technical stakeholders to create actors of a domain in an IDE graphically. Domain experts can then iteratively give behavior to the actors to send each other messages to accomplish a task. We will generate the corresponding source code and keep it up-to-date with the modeled actors.

V. Conclusion

"Moldable requirements" enables both technical and non-technical stakeholders to participate in requirements engineering and modeling processes by providing them appropriate and engaging interaction possibilities within an IDE. This approach and corresponding prototype implementation will allow researchers to think of issues, such as traceability, from a different perspective and simplify the requirements management.

Acknowledgment

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Agile Software Assistance" (SNSF project no. 200020-181973, Feb. 1, 2019 - April 30, 2022).

References

[1] I. Sommerville, "Software engineering 9th edition," *ISBN-10*, vol. 137035152, p. 18, 2011.

[2] M. dos Santos Soares, J. Vrancken, and A. Verbraeck, "User requirements modeling and analysis of software-intensive systems," *Journal of Systems and Software*, vol. 84, no. 2, pp. 328–339, 2011.

[3] N. Bik, G. Lucassen, and S. Brinkkemper, "A reference method for user story requirements in agile systems development," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2017, pp. 292–298.

[4] JIRA. Tool website at https://www.atlassian.com/software/jira. Accessed: 2020-11-22.

[5] A. G. Sutcliffe, N. A. Maiden, S. Minocha, and D. Manuel, "Supporting scenario-based requirements engineering," *IEEE Transactions on software engineering*, vol. 24, no. 12, pp. 1072–1088, 1998.

[6] Cucumber. Tool website at https://cucumber.io/. Accessed: 2020-11-22.

[7] T. S. Da Silva, A. Martin, F. Maurer, and M. Silveira, "User-centered design and agile methods: a systematic review," in *2011 AGILE conference*. IEEE, 2011, pp. 77–86.

[8] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," *Information Systems Journal*, vol. 20, no. 5, pp. 449–480, 2010.

[9] J.-Y. Mao, K. Vredenburg, P. W. Smith, and T. Carey, "The state of user-centered design practice," *Communications of the ACM*, vol. 48, no. 3, pp. 105–109, 2005.

[10] M. Wynne, A. Hellesoy, and S. Tooke, *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2017.

[11] H. Gomaa and L. Kerschberg, "Domain modeling for software reuse and evolution," in *Proceedings Seventh International Workshop on Computer-Aided Software Engineering*. IEEE, 1995, pp. 162–171.

[12] S. Dimitrijević, J. Jovanović, and V. Devedžić, "A comparative study of software tools for user story management," *Information and Software Technology*, vol. 57, pp. 352–368, 2015.

[13] A. Z. Yang, D. A. da Costa, and Y. Zou, "Predicting co-changes between functionality specifications and source code in behavior driven development," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 534–544.

[14] F. Zampetti, A. Di Sorbo, C. A. Visaggio, G. Canfora, and M. Di Penta, "Demystifying the adoption of behavior-driven development in open source projects," *Information and Software Technology*, p. 106311, 2020.

[15] V. Viyović, M. Maksimović, and B. Perisić, "Sirius: A rapid development of dsm graphical editor," in *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*. IEEE, 2014, pp. 233–238.

[16] E. Bousse, T. Degueule, D. Vojtisek, T. Mayerhofer, J. Deantoni, and B. Combemale, "Execution framework of the gemoc studio (tool demo)," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, 2016, pp. 84–89.

[17] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis lectures on software engineering*, vol. 3, no. 1, pp. 1–207, 2017.

[18] J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," *IEEE software*, vol. 31, no. 3, pp. 79–85, 2013.

[19] A. Chis, *Moldable tools*. Lulu. com, 2016.