# 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 thesis, 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 entities 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 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]. User requirements are often written using natural language, domain-specific models, or informal models [3], and they document the needs of the end-users and other stakeholders. System requirements are derived from user requirements with a detailed description of what the system should do and are usually modeled using formal or semi-formal methods and languages. 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 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 particular RE activities and distinct target audiences, requirements scatter through many documents and are maintained in numerous formats. Therefore, facilitating collaboration in an agile way among stakeholders also means making the employed tools to interact with each other.

Imagine building an address book application that allows users to create and add contacts to an address book. Requirements for this application are maintained in different formats and in several tools, from high-level ones, such as epics in a platform like Jira,[1]down to concrete scenarios that describe various operations in a separate tool like Cucumber.[2] In addition to specifying requirements, the domain experts need to verify whether the requirements are accurately implemented. They achieve it either by running tests or by interacting with a full-functioning user interface (UI). Both behavior verification approaches have limitations, as testing merely asserts how much functionality is actually working by hiding other details, and building a sophisticated UI costs time and effort.

As a solution, researchers have proposed to specify requirements directly in an IDE. However, this

[1]"Jira", accessed November 6, 2020 at https://www.atlassian.com/software/jira

[2]"Cucumber", accessed November 6, 2020 at https://cucumber.io/

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 entities 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 as first-class entities. Additionally, they must also provide appropriate interfaces to create, manage, and link the corresponding requirements to the implementation.

"Moldable requirements" is an approach that can be implemented in any IDE. We suggest that requirements hierarchies, as well as their representations, must be adapted (or *molded*) to suit the application domain and project needs. Developers first create custom requirements hierarchies. A sample hierarchy of requirements to implement the address book example might involve creating high-level epics, system-specific use cases, user-centered user stories, and concrete 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 example, for the address book application, stakeholders can use buttons and forms to create and save epics, use cases, *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. Therefore, non-technical stakeholders can inspect a *contact* with a contact card representation built by the developers. Finally, as requirements are created as first-class entities in an IDE, it enables us to link them to the involved live domain objects. In summary, both requirements hierarchies and their representations are molded in an IDE to suit the application and project context. 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, behavior-driven development (BDD), and visual domain modeling, *etc.* 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. They are widely used in practice due to their nature and understandability. Often user story management tools are isolated from the development environment, which leads to traceability issues. A recent study compared five user story management tools to report how well they satisfy functional requirements, such as support for epic management and support for testing [4]. Some of the proposed tools offer IDE integration; however, the characteristics of the available integration are not studied in the existing research. A recent survey of 182 practitioners and 21 semi-structured follow-up interviews expressed a necessity for better user story management tools [5]. Similarly, BDD proposes non-technical stakeholders to describe application behavior in a constrained natural language format. The behavior is tested by linking the behavior specification to the implementation through an automatically generated glue code [6]. The success of BDD

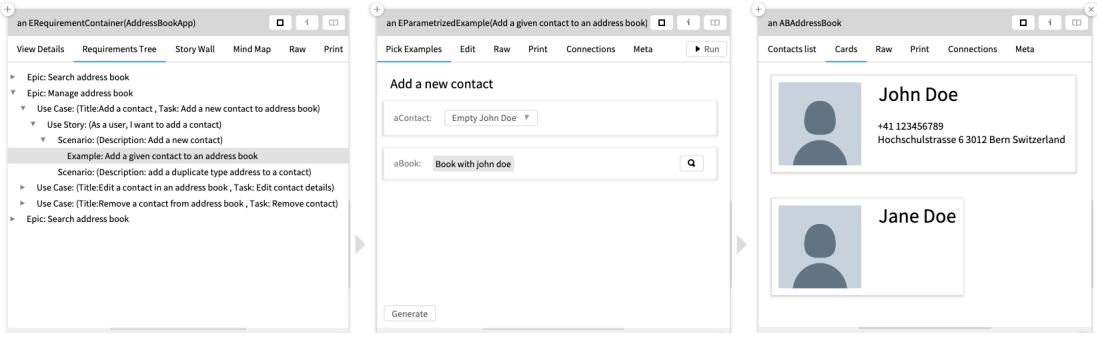


Fig. 1: Moldable requirements approach

depends heavily on the employed tools [7], [8]. 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 pass/fail status.

To tackle issues, such as ambiguities and inconsistencies, inherent with natural language specifications, requirements modeling approaches suggest using formal notations. There is an extensive support for requirements and domain modeling in IDEs [9], [10]. 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 underlying implementation technology, which facilitates communication between team members [11]. However, such approaches have received limited appreciation in practice due to extensive required training and laborious efforts for specifying detailed models [12].

## III. Moldable requirements

The term "moldable requirements" extends the concept of "moldable development [13]" to refer to an environment that enables distinct stakeholders to specify requirements at different levels of detail and with representations specific to a particular application domain. 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.

In the left-hand side window of Figure 1, a user explores a requirements hierarchy for the toy address book example, displayed in terms of epics, use cases, user stories, and scenarios. This hierarchy and the corresponding tree representation is custom created by developers. This view enables other stakeholders to effortlessly navigate between requirements, as well as to gain a general overview of the existing ones. The custom and context-aware graphical widgets in the middle window enable non-technical stakeholders to provide input parameters to a scenario and verify the output upon running. Here, a user selects an *address book* and a *contact* to be added. The last window shows the resulting *address book* object with a visual representation custom crafted by a developer. Such domain-specific representation enables non-technical stakeholders to inspect the details of the domain objects more efficiently compared to a test passed/failed status. Also, it eliminates the necessity to build a full-functioning UI to verify the application behavior. Figure 2 illustrates 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 epics, use cases, and user stories. In other words, it enables non-technical stake-

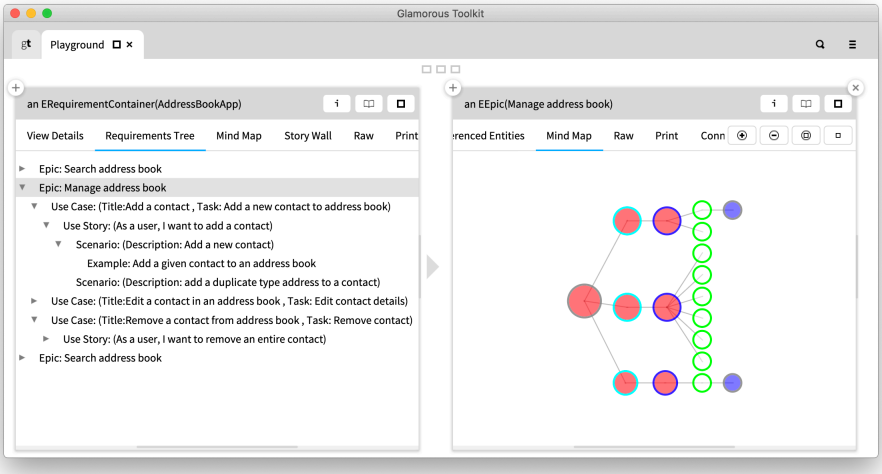


Fig. 2: Moldable requirements- multiple views

holders to create and access requirements without any programming overhead.

### A. *Current and planned contributions*

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

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

*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 despite the recent growing adoption of practices enabling non-technical stakeholders to write natural language specifications, 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 passed/-failed 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.

*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 [7], [8]. We conduct a study that takes a closer look at the characteristics of the glue code provided by the BDD tools in these projects.

*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 RE artifacts and discuss their characteristics, such as format, nature, phases of origin, and usage. We observed that several artifacts, *e.g.,* story wall, act as containers

for other artifacts, *e.g.,* story cards. Similarly, 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 an approach, wherein we model a selection of artifacts in an IDE.

*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. This project reviews a selection of user story management tools to uncover their limitations regarding support in IDEs. A prototype implementation demonstrates 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. This project studies the characteristics of a selection of graphical modeling tools. In particular, we take a comprehensive look at their support in an IDE and bi-directional change propagation mechanism they provide. We present an advanced prototype implementation of an approach that enables non-technical stakeholders to graphically create actors of a domain in an IDE. The actors are then iteratively given behavior to send each other messages to accomplish a task. The corresponding code is automatically generated and kept up-to-date with the modeled actors.

## IV. 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] M. Luisa, F. Mariangela, and N. I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, no. 1, pp. 40–56, 2004.

[4] 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.

[5] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, "The use and effectiveness of user stories in practice," in *International working conference on requirements engineering: Foundation for software quality*. Springer, 2016, pp. 205–222.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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