RM2PT: Requirements Validation through Automatic Prototyping

Yilong Yang*, Wei Ke‡, Xiaoshan Li*
*Faculty of Science and Technology, University of Macau, Macau
‡Macao Polytechnic Institute, Macau
Email: yylonly@gmail.com, wke@ipm.edu.mo, xsl@umac.mo

***Abstract*—Prototyping is an effective and efficient way of requirements validation to avoid introducing errors in the early stage of software development. Our previous work presents a tool RM2PT to automatically generate prototypes from requirements models. The stakeholders can easily check whether the requirements reflect their real needs by investigating the executions of use cases in the generated prototypes. However, the conflict and contradictory of the requirements are hard to be discovered. In this paper, we enhance RM2PT by introducing consistency checking and state observations in the generated prototypes. Requirements inconsistency can be automatically detected and further fixed through carefully analyzing the contracts of system operations and system state observations. We have conducted four case studies with over 50 use cases. The experimental result shows that 107 requirements inconsistency are founded in requirements validations. Overall, the result is satisfiable, and the enhanced RM2PT can be further applied to the software industry for requirements validation.**

The tool can be downloaded at http://rm2pt.mydreamy.net, and a demo video casting its features is at https://youtu.be/Y7GNa57WGfA

***Index Terms*—Requirements, Requirements Validation, Consistency Checking, Prototype, Prototyping**

Rapid prototyping is an effective approach to requirements validation to demonstrate concepts, discover requirements errors and find possible fixing solutions [1]. In practice, it is very desirable to generate prototypes directly from requirements automatically with a CASE tool. However, the state-of-the-art tools still have long distances to reach the goal [2]. Our previous work [3] presents a tool RM2PT to automatically generate prototypes from requirements models. The stakeholders can easily check whether the requirements reflect their real needs by investigating the executions of use cases in the generated prototypes. However, the conflict and contradictory of the requirements (especially between the contracts of system operations and invariants) are hard to be discovered only depending on investigating the execution of use cases.

In this paper, we present a tool RM2PT, which enhances our previous work by introducing consistency checking and state observation in the generated prototypes. Requirements inconsistency can be automatically detected and further fixed by observing system states with analyzing the contracts of system operations. The remainder of this paper is organized as follows: Section 2 presents the RM2PT features. Section 3 presents the evaluation results in the four case studies. Section 4 and 5 discuss the related tools and conclude this paper.

I. RM2PT Features

A. Automatic Prototyping

RM2PT can automatically generate prototypes from a requirements model, which contains a use case diagram, a conceptual class diagram (no operations in the classes) with class invariants, system sequence diagrams (system events only between actors and systems) for use cases, and the contracts of their system operations formally specified by a pair of pre- and post-conditions in OCL (Object Constraint Language).

B. Validity Checking

Validity checking focuses on checking whether the requirements reflect the real needs of stakeholders. The generated prototypes provide a function panel for use case executions. The stakeholders can pick up a system operation of a use case, type the input parameters, and click the execution button to check whether the system returns the expected results.

C. Consistency Checking

Consistency checking is to examine whether requirements models contain conflict and contradictory inside or between different stakeholders during requirements validation and evolution [4]. RM2PT can not only detect the syntax inconsistency of requirements models but also can detect the semantic inconsistency between the contracts (pre- and post-conditions) of system operations and invariants. In details, three kinds of requirements errors can make conflicts:

- **Pre-condition Errors** The execution of system operation containing pre-condition errors may lead the system to an unexpected state that violates system invariants.
- **Post-condition Errors** Post-condition errors can also lead the system to an unexpected state that violates system invariants and the pre-condition of the next system operations under the same use case.
- **Invariants Errors** If there is no error in pre- and post-condition of the contract, the invariant will not satisfy when it contains inappropriate constraints.

To detect the conflicts, the generated prototypes of RM2PT will check the corresponding contracts and all related invariants before and after the executions of system operations. If any requirements error makes the system into an unexpected state, the corresponding pre- and post-condition, and invariant

panels will be automatically marked as the red from green color in the generated prototype.

D. State Observation

The generated prototypes of RM2PT can automatically detect the inconsistency, but locating and fixing errors require more efforts. RM2PT provides the mechanism of observing the current state of the objects in the prototype to help developers to locate and fix the errors. For example, when the generated prototype indicates that a correct invariant is violated after the execution of system operation. The user can switch to the state observation panel to check whether the current states of objects are excepted. If not, the requirement errors must be in the pre-condition if the post-condition is correct.

II. Evaluation

We use four case studies to demonstrate the validity and capacity of the proposed approach for requirements validation. Those case studies are widely used systems in our daily life: supermarket management system (CoCoME), Library Management System (LibMS), Automated Teller Machine (ATM), and Loan Processing System (LoanPS). During the three-round requirements modeling, prototype generation, and requirements validation on those four case studies, we found **107** requirements errors, which includes 31 errors in the pre-condition, 68 errors in the post-condition, and 8 errors in the invariant of the contracts. More details can be found at GitHub[1].

III. Related Work

Test-case generation is the approach of requirements validation. CosTest [5] can automatically generate test cases and test reports for requirements validation from an fUML model compensated with Action Language Alf. fUML contains a class diagram and activity diagrams, the details of implementation are specified by Alf. That means CosTest not only need requirements specifications but also a design, which contains how to encapsulate the system operations into the classes, and the implementations of the system operations. Besides, test-case based validation is friendly for developers but not customers and clients, that makes it hard for them to validate their requirements through CosTest in practice.

The animation [6] is another important technique for requirements validation. The paper [7] presents a tool for interactively validating requirements through animation. It takes BPMN as input and generates a mock-up user interface prototype. The paper [8] presents a web animation tool for requirements validation through exploring goals and scenarios, in which it uses linear temporal logic to express goals, and XML to define state transitions and UI components. TestME-Req [9] can automatically generate mock-up user interface as well as abstract test cases from requirements description. It allows multiple stakeholders to collaboratively validate the same set of requirements.

[1]https://github.com/RM2PT/CaseStudies

Compared with our work, the prototypes from their tools are only mock-up, which do not contain the implementation of the details of system operations. That means only the coarse-grained requirements validation can be done through basic animations, validity and consistency checking are not included. Moreover, the mock-up prototype will be throw-away after validation. The prototypes generated from our tools are evolutionary, they embedded architecture patterns and design patterns in Java EE and .NET enterprise system. That makes the generated prototypes from our tools easier evolving to be the practical software systems without too much cost.

IV. Conclusion

This paper presents a tool for requirements validation through automatic prototyping. Requirements errors can be found and fixed through executing of use cases in the prototype with the new proposed function: consistency checking and state observation. Four cases studies have been investigated, and the experiment result is satisfactory that the 107 requirements errors are founded during validation of the features of RM2PT. In the future, we will continue working on the fault localization and fixing of requirements validation, especially for reducing the human efforts in this process. Hopefully, it can benefit the software industry in requirements engineering.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (NSFC) and the Macao Science and Technology Development Fund Joint Scientific Research Project (No. 0001/2018/AFJ) and NSFC project (No. 61562011).

References

[1] F. Kordon and Luqi, "An introduction to rapid system prototyping," *IEEE Transactions on Software Engineering*, vol. 28, no. 9, pp. 817–821, Sep. 2002.

[2] F. Ciccozzi, I. Malavolta, and B. Selic, "Execution of UML models: a systematic review of research and practice," *Software and Systems Modeling*, vol. 18, no. 3, pp. 2313–2360, Jun. 2019.

[3] Y. Yang, X. Li, Z. Liu, and W. Ke, "RM2PT: A tool for automated prototype generation from requirements model," in *Proceedings of the 41th International Conference on Software Engineering: Companion Proceedings (ICSE'19)*, May. 2019, pp. 59–62.

[4] I. Hadar and A. Zamansky, "Cognitive factors in inconsistency management," in *Proceedings of the 23th IEEE International Requirements Engineering Conference (RE'15)*, Aug. 2015, pp. 226–229.

[5] M. F. Granda, N. Condori-Fernández, T. E. J. Vos, and O. Pastor, "CoSTest: A tool for validation of requirements at model level," in *Proceedings of IEEE 25th International Requirements Engineering Conference (RE'17)*, Sep. 2017, pp. 464–467.

[6] A. Gemino, "Empirical comparisons of animation and narration in requirements validation," *Requirements Engineering*, vol. 9, no. 3, pp. 153–168, Aug. 2004.

[7] G. Gabrysiak, H. Giese, and A. Seibel, "Deriving behavior of multi-user processes from interactive requirements validation," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE'10)*, Sep. 2010, pp. 355–356.

[8] S. Uchitel, R. Chatley, J. Kramer, and J. Magee, "Fluent-based animation: exploiting the relation between goals and scenarios for requirements validation," in *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04)*, Sep. 2004, pp. 208–217.

[9] N. A. Moketar, M. Kamalrudin, S. Sidek, M. Robinson, and J. Grundy, "An automated collaborative requirements engineering tool for better validation of requirements," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16)*, Sep. 2016, pp. 864–869.