UDC 004.4:004.738.5(045)

O.V. Chebanyuk,
(National Aviation University, Ukraine)
Abdel-Badeeh M. Salem
(Ain Shams University, Egypt)

Software models' refinement in AGILE approach. Review and Challenges.

This article represents a review of approaches to software models refinement. Then complex recommendations for raising effectiveness of software development live cycle processes are proposed. The aim of refinement operation is to improve software models' quality in AGILE approach.

"Why is it that some software engineers and computer scientists are able to produce clear, elegant designs and programs, while others cannot? Is it possible to improve these skills through education and training?"
J. Kramer, Japan (J. Kramer, 2007)

Introduction to software models refinement

Refinement is a variant of horizontal model to model transformation operation (Khif et al., 2018). This idea is explained by the fact that initial and resulting models have the same level of detailization (Brambilla et al., 2012).

Refinement procedure allows obtaining of quality software models that answer to Model-Driven Engineering approach.

Following OMG (Object Management Group) MOF (MetaObject Facility) recommendations to perform a refinement operation we conside refinement metamodeling stack proposing high-level and low-level procedures.

Great contribution in development of refinement approaches, namely high-level and low-level procedures is made by authors of papers (Khif et al., 2018), (Brambilla et al,. 2012), (Hinkel, G., 2018 et al., 2018), (Kramer J., 2007) , (Dhaou, F., 2016) and many others.

The contribution of this paper is a representation of systematized review of software models' refinement approaches and grounding the foundations of using analytical tools to design software models refinement language.

Review of refinement approaches

Review of approaches deals with analytical foundation of refinement (high-level procedures).

The foundation of refinement approach was proposed in book (Back and Von Wright, 1998). Authors explain the refinement calculus as a framework for reasoning about correctness and refinement of programs. Each refinement step is required to preserve the correctness of the previous version of the program. Basic refinement rules for refinements are kinds of program derivation in detail, looking at specification statements and their implementation, how to construct recursive and

iterative program statements, and how to use procedures in program derivations. The refinement calculus is an extension of Dijkstra's weakest precondition calculus, where program statements are modeled as predicate transformers extend the traditional interpretation of predicate transformers as executable program statements.

Authors have to reason about properties of functions, predicates and sets, relations, and program statements described as predicate transformers. These different entities are in fact mathematically all very similar, and they form the refinement calculus hierarchy.

Other operation that deals with code refinement is refactoring. Traditional refactoring (Fowler, 1999) improves the code structures while preserving the external functionality.

Other papers consider refinement related to one type of UML diagrams.

Paper (Hinkel, G., 2018) proposes NMeta language for software models' refinement.

This language is an extension of OCL and contains several new expressions for checking consistency of metamodels. Authors also explain that there is a software for class diagrams refinement. But considering the fact that OCL works for class diagram, the refinement approach can be used only for them. Other important thing is that authors allow to refine relationship between two links connecting classes. This fact leaves unclear as well as a principle how refinement framework implements NMeta constraint language functions. Authors give only short descriptions how both framework and code generation tools are functioning.

Paper (Nieto et al., 2011) proposes a semantics for association redefinition and using a similar notation. However, they also implement refinements through a constraint of the more general reference and therefore do not inherit type-system guarantees.

In the paper (Hinkel, G., et al, 2018) authors proposed a formal definition of refinements and structural decomposition, how they can be implemented in a meta-metamodel and how a code generator can be designed to ensure them through type system guarantees. This can make many validation constraints. In this case refinement allows to set more exact relations between class diagram elements. An approach, proposed in paper (Hinkel, G., et al, 2018), describes a semantics by extending the semantics of the refinement information to structural decomposition.

An approach that deals with refinement of sequence diagrams, while preserving required behaviours and deals correctly with guards is proposed in paper (Dhaou, F., 2016). Authors formalize the refinement relation favourable to an incremental development, it is based on existing ones. Finally, a generic implementation with event refinement for checking of correctness of refinement relation is proposed.

Conclusion from the review Investigations of refinement operations performed today are concentrated on three main refinement aspects namely analytical approaches of software models' refinement, behavioral software models' refinement, and static software models of architectural solutions refinement. Some of these researches include analytical refinement base, some of them use graphical notations at metalevel. But plenty of them are focused on having some reference templates that are used to

compare software models with them. After such a comparison it is proposed to modify previously designed software model.

Conclusion: Mostly such refinement operations are performed on refinement patterns, expressed by predicate expressions or other constraints notations (set theory rules or OCL constrains). Drawback of many software modeling processing tools and techniques is that they represent results of refinement in memory. To improve the refinement it is necessary to consider detailed representation of analytical tools for software models' refinement operation as it is shown in the figure 1. Detailed systematization of analytical tools involved to software models' processing is represented in paper (Chebanyuk and Markov, 2016).

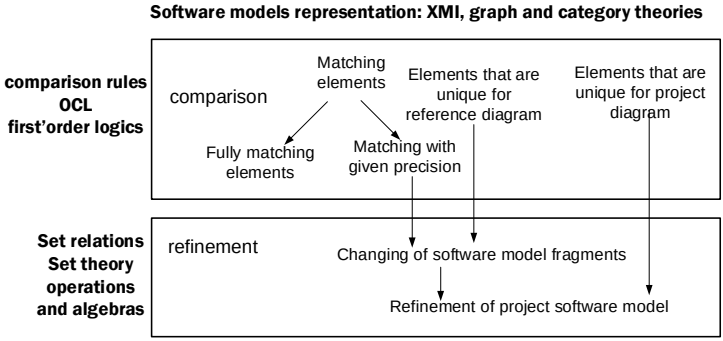

Fig. 1. Detailed representation of analytical tools for software models' refinement operation

Further research: As software model comparison is the first step of refinement, it is necessary to develop smart comparison tool, as it is described in a paper (Chebanyuk and Badeeh, 2018) to visualize changes performed while software models refinement is performed.

References

1. Back, R.-J. and Von Wright, J. (1998). *Refinement calculus: a systematic introduction.* springer Heidelberg
2. Brambilla, Marco; Cabot, Jordi; Wimmer, Manuel. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 2012, vol. 1, no 1, p. 1-182.
3. O. Chebanyuk, Abdel-Badeeh M. Salem Formal Foundation, Approach, and Smart Tool for Software Models' Comparison. Egyptian Computer Science Journal Volume 42, Number 4-2018 p. 89-102

4. Chebanyuk E., Markov K. Model of problem domain “Model-driven architecture formal methods and approaches.” International Journal “Information Content and Processing”, 2016, ISSN 2367-5128 (printed), ISSN 2367-5152, P 203-222.

5. Dhaou, F., Mouakher, I., Attiogbé, C. and Bsaies, K. Refinement of UML2.0 Sequence Diagrams for Distributed Systems. DOI: 10.5220/0006005403100318 In Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA, pages 310-318 ISBN: 978-989-758-194-6

6. Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

   Hinkel, G., Busch, K. and Heinrich, R. Refinements and Structural Decompositions in Generated Code. DOI: 10.5220/0006549403030310 In Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018), pages 303-310 ISBN: 978-989-758-283-7

7. Khlif I., Hadj Kacem M., Hadj Kacem A. and Drira K. A UML-based Approach for Multi-scale Software Architectures. DOI: 10.5220/0005380403740381 In Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS-2015), pages 374-381 ISBN: 978-989-758-097-0

8. Kramer, J. (2007). Is abstraction the key to computing? In Communications of the ACM, Vol. 50 Issue 4, pp.36-42