Using Reverse Semantic Traceability for Quality Control in Agile MSF-based Projects

Konstantin Zhereb[1], Vladimir Pavlov[1], Anatoliy Doroshenko[1], and Victor Sergienko[1]
[1]International Software & Productivity Engineering Institute (INTSPEI)
{ zhereb, vlpavlov, doroshenko, sergienko}@intspei.com

Abstract

Reverse Semantic Traceability (RST) is a quality control method that allows minimizing inconsistencies between inputs and outputs of every step in a software development process. For each step, before proceeding to the subsequent ones, the current inputs are restored (reverse engineered) from the current outputs, and compared to the original versions of inputs. If they are semantically different, then some corrective actions are required to eliminate the inconsistency.

Previously RST was successfully used within projects that employed "formal" methodologies. This paper describes experience of integrating RST into one of agile methodologies (Microsoft Solutions Framework for Agile Software Development). The paper also provides a case-study of using the new combined methodology in a small software development project.

***Keywords**: INTSPEI P-Modeling Framework, agile processes, Microsoft Solutions Framework (MSF), quality control.*

1. Introduction

Most currently used Software Development Life Cycles (SDLCs) have the same drawbacks in common. For large projects the most important decisions and the most expensive mistakes are done at the very beginning of the project, and then as the development moves forward the cost of mistakes goes down. This idea is illustrated on Fig. 1 [1]. At the same time, the amount of quality control activities is minimal at the beginning of the project but is increasing as development progresses. Hence, important analysis and design mistakes are usually discovered on the latest phases of the development process which leads to expensive rework.

There are two ways to deal with this problem: to shorten development iterations and make them more frequent, and to incorporate additional quality control techniques to identify analysis and design mistakes when they are introduced – not on the latest phases of development. The first approach became extremely popular recently as agile methods spread over the world [2]. Short iterations allow frequent customer feedback; also automated unit testing can be applied early in the project. However, for large projects the approach of shortening iterations does not work well; there is a need for quality control methods that utilize the second approach. Such methods include various forms of software reviews [9], as well as recently developed Reverse Semantic Traceability – a part of INTSPEI P-Modeling Framework [3].

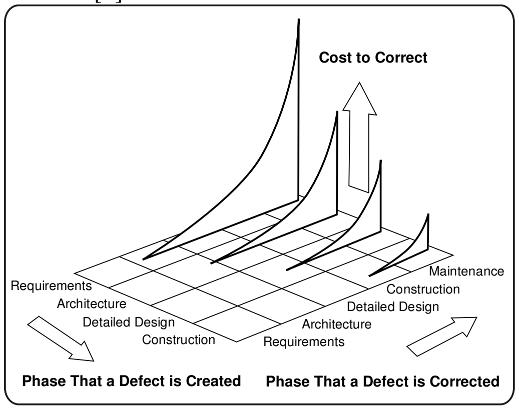


Figure 1. Software defects costs

INTSPEI P-Modeling Framework was first applied in large projects and got successful results described in [5]. Initial feedback from early adopters of INTSPEI P-Modeling Framework suggested that it could be used in agile projects as well. Therefore, we have created an integrated process that combined P-Modeling Framework and one of agile processes – Microsoft Solutions Framework for Agile Software Development (MSF Agile) [16]. Paper [6] describes our experience of integrating P-Modeling Framework with MSF Agile, and also provides technical description of the resulted integrated process. In this paper, we focus on methodological aspects of applying P-Modeling Framework in agile processes (exemplified by MSF Agile), rather than on technical details. This paper also contains a case study that demonstrates how to use Reverse Semantic Traceability together with agile methodologies.

The rest of the article is organized as follows. In Section 2, we present an overview of Reverse Semantic Traceability and compare it with other similar approaches. Section 3 outlines the integration of P-Modeling Framework with an agile process – MSF Agile. Section 4 describes a small case study of using RST (and P-Modeling) in agile project. Finally, Section 5 summarizes conclusions and presents the directions for future research.

2. Reverse Semantic Traceability

This section provides brief description of a new quality control technique, Reverse Semantic Traceability, which is the most important part of the INTSPEI P-Modeling Framework. Also we compare RST with other similar approaches found in literature.

2.1. The idea of the method

Reverse Semantic Traceability (RST) is a quality control method that allows testing consistency between inputs and outputs of every process step. For each step, before proceeding to the subsequent ones, the current inputs are restored (reverse engineered) from the current outputs, and compared to the original versions of inputs. If they are semantically different, then the step has to be repeated more precisely to eliminate this ambiguous understanding.

The key word in the name of this method is "Semantic" because the original and restored versions of an artifact are to be compared semantically, with a focus on the "meaning" of this artifact, not on particular notions used in it. Hence, the reverse engineering and the evaluation of the semantic difference must be performed by human. Based on this evaluation, a quantitative value can be assigned to each traceability relation between inputs and outputs.

The previous research related to Reverse Semantic Traceability focused on core ideas behind RST [3], as well as its application in education [4] and in large industrial projects [5]. In particular, paper [5] demonstrated that Reverse Semantic Traceability method could be used successfully for large projects that utilize formalized methods such as RUP or MSF CMMI.

INTSPEI P-Modeling Framework is not discussed here in detail. More detailed discussion of this methodology can be found in [3-8].

2.2. Similar techniques

RST improves existing quality control procedures, such as software inspections [9], by utilizing elements of reverse engineering and traceability management approaches. In this section we compare RST with similar techniques.

RST uses the procedure similar to software reviews and inspections [9-11], when reverse engineers read the "text" of the translated artifact to restore the original artifact. Dunsmore et al. propose the similar technique for reading object-oriented code [10]. However, the distinction of RST is that the reviewer is not allowed to be familiar with the input artifacts – he should restore them from the outputs. RST also adds one more step to the review process, namely comparing restored and original artifact. It helps to ensure that the output artifacts do not conflict previously created input artifacts (e.g. architecture is created according to the requirements or bug fix was performed according to bug description). This additional step also increases the efficiency of the review process, as reviewer (reverse engineer) has to examine the translated artifact more precisely in order to restore the original artifact.

There are also many solutions for capturing traceability relations between project artifacts on different stages of SDLC [12-15]. Traceability is used to establish links between requirements and source code fragments implementing these requirements, as well as to estimate effect of changes in requirements on source code [12]. Research in this area concentrates on metamodels for traceability process [13], usage scenarios of traceability [13, 14], automatic creation of traceability links [15]. However, most traceability solutions provide facilities only for establishing links between artifacts and not for estimating quality of these links. RST approach enhances traceability solutions by assigning numeric quality values to intermediate traceability links (e.g., from requirements to design).

3. Integration with MSF Agile

INTSPEI P-Modeling Framework is an add-on to existing methodologies, not a standalone process. Currently, INTSPEI P-Modeling Framework integrates both with agile and formal processes. The former include Microsoft Solutions Framework for Agile Software Development [16] and Open Unified Process [17], while the latter include IBM Rational Unified Process [18] and Microsoft Solutions Framework for CMMI Process Improvement [16]. In this paper we focus on integration with agile processes, using the P-Modeling Framework Integrated with MSF Agile as an example. This paper presents only an overview of P-Modeling Framework Integrated with MSF Agile; more detailed technical description is provided in [6].

3.1. Elements of P-Modeling in MSF Agile

The core elements of the INTSPEI P-Modeling Framework are the Reverse Semantic Traceability and the Speechless Modeling techniques. Both of them were incorporated into the MSF Agile lifecycle. The P-Modeling Framework Integrated with MSF Agile contains detailed descriptions of these techniques and step-by-step instructions for team members. It also describes how Reverse Semantic Traceability and Speechless Modeling work together with other MSF Agile activities.

P-Modeling Framework integrated with MSF Agile suggests performing the following RST activities:

- Perform RST for Scenario;
- Perform RST for Solution Architecture;
- Perform RST for Development Task Implementation;
- Perform RST for Database Task Implementation;
- Perform RST for Bug Fix;
- Perform RST for Scenario Test Cases;
- Perform RST for Quality of Service Requirement Test Cases.

The RST activities are connected to MSF Agile process. When one of the important work products is created according to MSF Agile guidance, P-Modeling Framework recommends performing an RST session to verify that no information was lost or misinterpreted during its creation. The typical RST session consists of the following steps: Preparation, Reverse Engineering Step, Expert Assessment and Making the Decision.

The participants of an RST session assume a set of RST-specific roles, which are active only during a single RST session. P-Modeling Framework adds three new roles to the set of MSF Agile roles: Artifact Owner, Reverse Engineer and Expert, and also uses one of the MSF roles, Project Manager. The RST roles are typically combined with the MSF roles. For example, the person performing the MSF Agile role "Architect" will also perform the RST role "Artifact Owner" during the RST session for the solution architecture. The same person can fulfill different roles in different RST sessions; furthermore, this practice is encouraged in order to increase the understanding of the P-Modeling Framework.

The results of the RST session are captured in specific work products – RST Session Report and RST Expert Assessment. The RST Session Report contains all information about the session, including the date, participants, original and restored artifacts and the final decision. The RST Expert Assessment form captures the result of experts' meeting – their assessment of quality value and their comments. These work products are used to communicate the results of the RST Session to the team.

Based on the results of the RST session, the Project Manager makes one of the following decisions:

1. The quality of the artifacts is sufficient and the development may proceed to the next phase.
2. Rework of output artifacts is needed in order to eliminate defects and information loss.
3. Corrections to both input and output artifacts are needed in order to eliminate misunderstandings
4. One more RST session after rework of the artifacts is required.

While the Reverse Semantic Traceability can be applied to all work products, this would create a significant overhead. Therefore, P-Modeling Framework recommends prioritizing the work products and performing RST only for the most significant of them. The prioritizing is performed be the Project Manager during iteration planning (at the beginning of each iteration). The artifacts are prioritized according to multiple criteria, including their contribution to the quality of the final product, severity of possible defects and availability of other quality control methods. The results of this activity are recorded in the RST Rank Table.

4. A case study

In this section we describe a small case study performed to investigate the usage of P-Modeling Framework in agile project.

4.1. Project description

We applied P-Modeling Framework integrated with MSF Agile in a pilot project. The project consisted in creating an application for simulation of Conway's Life game [19]. The project lasted for 2 months and included 3 part-time student developers.

During the project, the team followed the MSF Agile process, as described in MSF Process Guidance version 4.1 [16]. The project consisted of four 2-week iterations. The first iteration corresponded to MSF Envision and Plan tracks – the team created vision document, collected requirements and outlined initial design. Two intermediate iterations were spent on actual development and testing of the product (Build track). The last iteration combined activities from Build and Stabilize tracks.

The team started using P-Modeling Framework from the beginning of the project. On the first days of the first iteration, the planning of RST activities was performed. Each RST session requires at least one external participant who is not familiar with the project details, and the Project Manager should look for such candidates in advance. Therefore, the Project Manager should understand how many RST sessions are expected on each iteration and for what artifacts.

In our project, the team noticed that RST activities would differ significantly in the first iteration and in all subsequent iterations. During the first iteration, the most important artifacts are vision document and high-level design. Performing Reverse Semantic Traceability sessions to verify these artifacts constitutes a valuable addition to MSF process. Their quality could not be efficiently verified by other means – at least until the implementation is created in the next iterations. In contrast, for all subsequent iterations the focus of RST activities shifts to verifying multiple smaller artifacts: implementations of development tasks, bug fixes and test cases. In these iterations, Reverse Semantic Traceability complements other quality control methods, such as unit testing and functional testing.

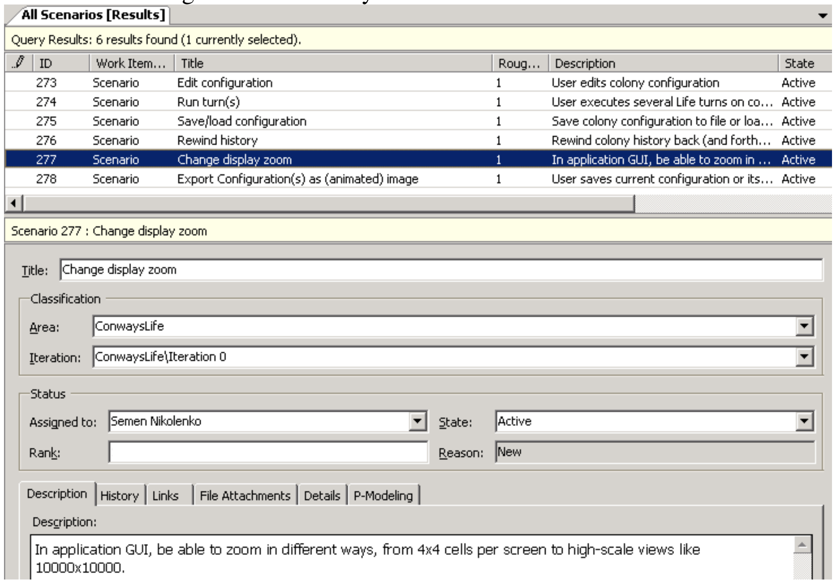
P-Modeling Framework integrated with MSF Agile proposes to verify vision statement by restoring it from scenarios (the process is described in "Perform RST for Scenario" activity); the high-level design is verified by restoring the requirements from the design. When the team planned RST activities, they decided that performing RST session for the design will be more valuable. The vision for the project was quite simple, and the team expected that it should not contain any defects. The design, on the other hand, could contain a number of defects, because the student who created it had little experience in design; also miscommunications were possible, as requirements and design were created by different team members. Also the team decided that the importance of design was greater than that of vision.

This paper concentrates on Reverse Semantic Traceability activities that were performed in the first iteration only.

4.2. RST session for Design

The most important of RST activities for the project was RST session for high-level design. The team created the design of the application in the first iteration. During RST session, reverse engineers were assigned to restore the requirements (in form of scenarios) from the design. The reverse engineers were not familiar with the project before the RST session. When the requirements were restored, a team of experts compared the restored and original versions of the requirements and expressed their comments.

The initial requirements were split into functional and non-functional (called scenarios and quality of service requirements in MSF Agile). The requirements were stored in Microsoft Team Foundation Server. Fig. 2 shows the scenarios identified by the Analyst. The main scenarios are: editing configuration, running simulation for one or more turns and saving or loading game configuration. Additional lower-priority scenarios included moving through simulation history, changing the size of the displayed configuration and exporting configurations as images. (The full text of the requirements is not included because of space limits.)



Query Results: 6 results found (1 currently selected).

ID	Work Item...	Title	Roug...	Description	State
273	Scenario	Edit configuration	1	User edits colony configuration	Active
274	Scenario	Run turn(s)	1	User executes several Life turns on co...	Active
275	Scenario	Save/load configuration	1	Save colony configuration to file or loa...	Active
276	Scenario	Rewind history	1	Rewind colony history back (and forth...	Active
277	Scenario	Change display zoom	1	In application GUI, be able to zoom in ...	Active
278	Scenario	Export Configuration(s) as (animated) image	1	User saves current configuration or its...	Active

Scenario 277 : Change display zoom

Title: Change display zoom

Classification
Area: ConwaysLife
Iteration: ConwaysLife\Iteration 0

Status
Assigned to: Semen Nikolenko
State: Active
Rank:
Reason: New

Description | History | Links | File Attachments | Details | P-Modeling

Description:
In application GUI, be able to zoom in different ways, from 4x4 cells per screen to high-scale views like 10000x10000.

Figure 2. Original scenarios

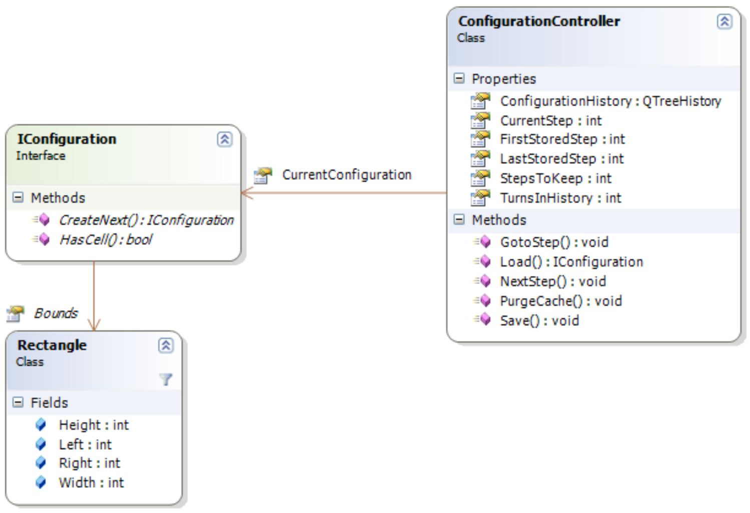


Figure 3. Core classes

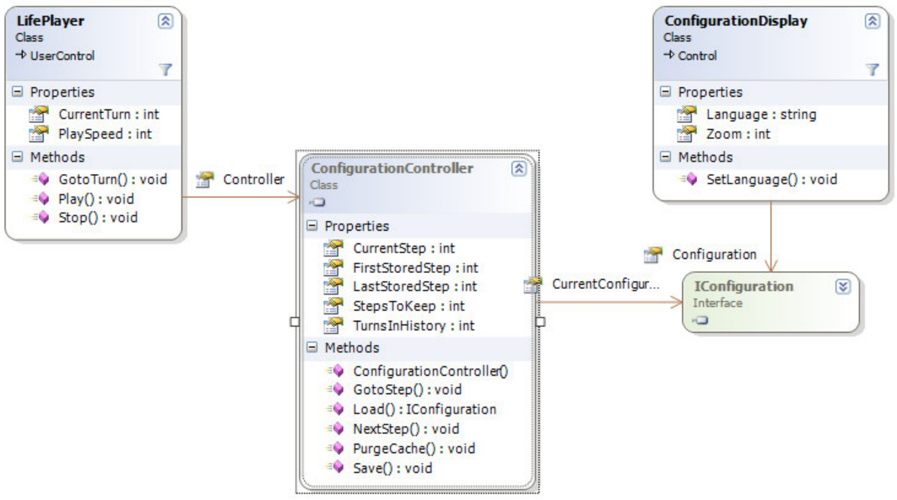


Figure 4. GUI classes

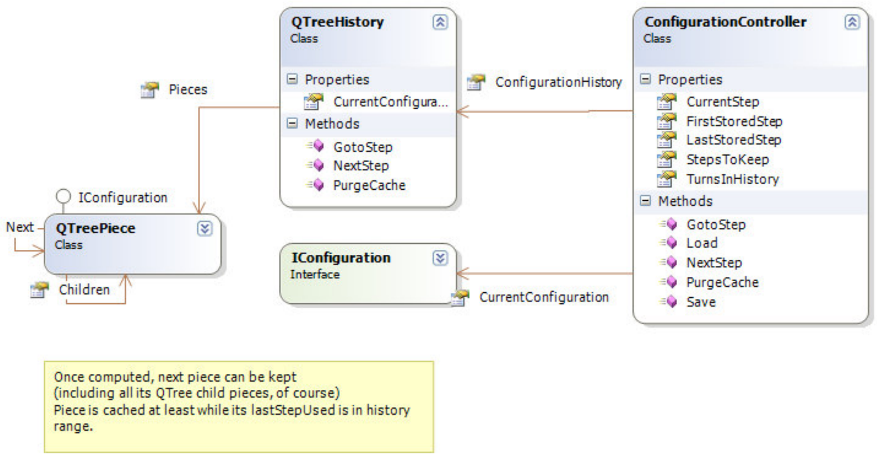


Figure 5. “Hashlife” algorithm classes

Based on the requirements found, the Architect developed the application design using Microsoft Visual Studio. The core classes are outlined on Fig. 3. The IConfiguration interface represents single colony configuration; ConfigurationController calculates next game step, stores the history and saves or loads a game state. The classes responsible for graphical user interface (ConfigurationDisplay and LifePlayer) are shown on Fig. 4. Fig. 5 presents classes related to "Hashlife" algorithm [19] that performs simulation.

When the diagrams representing design were ready, an RST session was performed to verify that the architecture meets the requirements. Reverse engineers were provided with the design and were asked to restore the requirements. The reverse engineers had no prior knowledge of any project-related activities; however, they did know the rules of the Life game. The duration of the RST session was restricted to 1 hour. The reverse engineers restored the following requirements (Fig. 6).

After the reverse engineers have restored the requirements, experts compared them with the original requirements. The process took about an hour: first 40 minutes were spent on reading documents by individual experts, followed by 20-minute discussion. The experts produced the following comments (Fig. 7).

Based on the results of the RST session, the team made the following rework decisions:

- Modify requirements – add missing scenario (Choose language)
- Clear up domain dictionary – use "Colony" instead of ambiguous "Configuration"
- Put more work into "Edit configuration" functionality– notable design changes;
- More detailed GUI classes design – minor design changes;

Scenario 1: User can control the game of "Life" by performing the following actions:

- Start the game using "Play" command
- Stop the game using "Stop" command
- Move to the given turn using "Go to Turn" command

Additional Requirements

- The system simulates each game step with the speed specified by "PlaySpeed:int" parameter
- The systems stores the history of game steps
- Hashlife implementation is used (http://en.wikipedia.org/wiki/Hashlife). The fast storage of simulation history is implied.
- There is an option to clear the game history.

Scenario 2: The player can change game display configuration. The player can set the display language.

Additional Requirements

- Display settings: language, zoom

Notes:

- User interface provides no possibility to change zoom level and simulation speed
- There is a mistake in Rectangle class: it should contain height, width, left, top, instead of height, width, left, right.

Figure 6. Restored requirements

1. Requirement not restored: Edit configuration;
2. Requirement not restored: Save/load configuration;
3. Requirement not restored: Rewind history;
4. Requirement not restored: Export Configuration(s) as animated image;
5. Extra requirement restored: Change language
6. Extra requirement restored: Clear history
7. User interface provides no possibility to change language
8. Note: What is "configuration"? Looks like "program configuration" and can be easily confused with it. I suggest naming it like "colony" or "board".

Figure 7. Expert comments

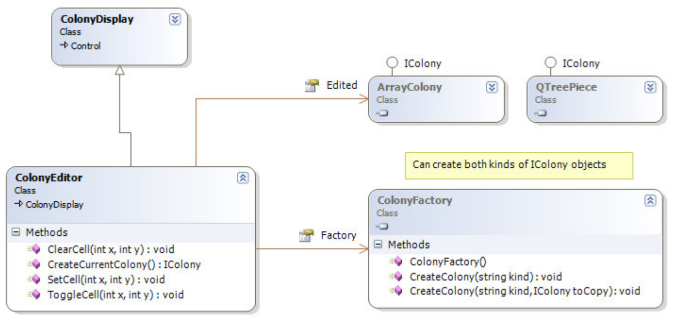

ColonyDisplay
Class
Control
ColonyEditor
Class
ColonyDisplay
Methods
ClearCell(int x, int y) : void
CreateCurrentColony() : IColony
SetCell(int x, int y) : void
ToggleCell(int x, int y) : void
Edited
Factory
IColony
ArrayColony
Class
IColony
QTreePiece
Class
Can create both kinds of IColony objects
ColonyFactory
Class
Methods
ColonyFactory()
CreateColony(string kind) : void
CreateColony(string kind, IColony toCopy): void

Figure 8. Changes in design after RST: a new diagram

Fig. 8 shows the new class diagram that was created as a part of improvements suggested by the RST session. It contains the classes related to the scenario that was initially omitted from design and hence not restored by reverse engineers ("Edit configuration"). Note that other diagrams have also changed; we don't include updated versions because of space limit.

4.3. Analysis

The RST session performed as a part of MSF Agile project demonstrated that Reverse Semantic Traceability can be a valuable quality control technique in a project based on agile methodologies. The most important usage scenario for RST is quality control at the beginning of the project, in the first iterations. Even in agile processes, there is a period of time at the project start when no production code is developed. The team focuses on establishing project vision and important architectural decisions. The typical quality control methods of agile processes – automated unit tests and customer feedback – are of limited value in the first iteration. However, the Reverse Semantic Traceability can verify that the first crucial artifacts are correct (or at least consistent).

The team members were excited to discover the method of discovering defects in project artifacts without actually testing the software (or even writing the code). The only other method they knew that could produce the comparable results was software review process. However, the feedback from the team shows that the RST method is more effective. One of the reverse engineers said, "I always thought that design review is a boring process. But the RST session was anything but boring. It was more like solving an exciting puzzle – trying to understand the reasons behind the design decisions … I believe that RST is more effective than traditional design review. It was the task to restore the unknown requirements that forced us to understand the design better and so find some subtle defects that would evade our attention during design review."

The RST session proved quite effective in terms of the resources spent and the outcome. The whole RST session required about 7 man-hours: 1 for preparations, 1 hour for two reverse engineers and 1 hour for 4 experts (most of this time was spent by people not directly involved in the project – reverse engineers and experts). However, the outcome of the session was quite significant. Both the requirements and the design have been improved, and some defects were eliminated that would require significant rework were they discovered during actual coding. For example, the "Edit colony (configuration)" scenario that was omitted from design required significant changes in user interface, as well as in core classes. When the improvements suggested by RST were made, the team noticed that one of important assumptions about core classes was wrong. Namely, the QTreePiece class (see Fig. 5) was assumed to be immutable, because of requirements of the "Hashlife" algorithm. However, the "Edit colony" scenario required that the colonies (implemented by QTreePiece class) could be modified. As a result, the team decided to add one more class representing a colony (ArrayColony) that was mutable. This change in design required about 2 hours. However, the potential cost of rework would be much greater had the wrong design been implemented in code. The team estimated that this defect (missing edit capability) would have been certainly found by the team; but its correction would require significant changes in user interface, core classes and unit tests. The rework would stop the progress of the entire team for a few working days.

The feedback from the case study participants (both project team and RST participants) suggested that Reverse Semantic Traceability effects were not limited to improving the quality of the artifacts that were used in RST session. The team members learned that the project artifacts are not created just because the process says to do so. The artifacts created by one team member will be used by another one, and the author should make the artifact understandable to its subsequent user. The person who created the design said, "I tried to create the design that would be correct and elegant, but I actually forget that it should be understandable. The reverse engineers said they had some problems understanding my diagrams; but if we did not perform RST, the same problems would hinder the progress of developers". Therefore, RST session ensured that artifacts that were created could be actually used in project. This corresponds to the agile principles stating that the artifacts that are not useful to the project should be avoided. RST helped to make project artifacts, including requirements and design, actually used in project, not just "write-only" bureaucratic burden.

5. Conclusions

We have described an application of Reverse Semantic Traceability technique for quality control in agile project and illustrated it by a case study. The case study has demonstrated that Reverse Semantic Traceability can be used in agile projects, and it provides a valuable addition to agile quality control methods. According to our observations, the usage of RST is the most important during the first iteration of agile project.

The future directions of research include applying P-Modeling Framework in industry projects based on agile process. Also we want to conduct more experiments applying Reverse Semantic Traceability to different kinds of artifacts in order to improve the RST process and create more detailed instructions on usage of P-Modeling in both agile and more formal processes.

6. References

[1] Steve McConnell: Upstream Decisions, Downstream Costs. Windows Tech Journal (1997) http://www.stevemcconnell.com/articles/art08.htm

[2] Manifesto for Agile Software Development http://www.agilemanifesto.org

[3] Vladimir L Pavlov, Stanislav Busygin, Nikita Boyko, Alexander Babich, "Is There Still a Room For Programmers' Productivity Improvement?", Proceedings of the 5th East-West Design and Test Symposium (EWDTS'07), 2007, pp. 146-151

[4] Pavlov, Vladimir; Boyko, Nikita; Babich, Alexander; Kuchaiev, Olexii; Busygin, Stanislav., Applying Pantomime and Reverse Engineering Techniques in Software Engineering Education, Proc. 37th ASEE/IEEE Frontiers in Education Conference, Oct. 10 – 13, 2007, Milwaukee, WI. IEEE, 2007, pp. TIE1-TIE5.

[5] Pavlov, Vladimir; Boyko, Nikita; Babich, Alexander; "First Experience of Using INTSPEI P Modeling Framework in Software Development Projects", Problems in Programming, Issue 2, May 2007, pp. 68-75.

[6] V. Pavlov, A. Doroshenko, T. Taganskaya, K. Zhereb, N Boyko, An Experience of Integrating INTSPEI P-Modeling Framework with Microsoft Solutions Framework for Agile Software Development, 2007 (accepted for publication in Proc. IASTED Int. Conf. Software Engineering, 2008).

[7] Pavlov, Vladimir; Yatsenko, Anton "'The Babel Experiment': An Advanced Pantomime-based Training in OOA&OOD with UML", 36th 'ACM Technical Symposium on Computer Science Education', February 25, 2005.

[8] INTSPEI P-Modeling Framework Whitepaper, INTSPEI, http://www.intspei.com

[9] Fagan, M.E., Design and Code Inspections to Reduce Errors in Program Development. IBM Syst. J., Vol. 15, No. 3, 1976, pp. 181-211.

[10] A. Dunsmore, M. Roper, and M. Wood, "Systematic Object-Oriented Inspection—An Empirical Study," Proc. 23rd Int'l Conf. Software Eng. (ICSE '01), pp. 135-144, May 2001.

[11] Wood, M., Roper, M., Brooks, A., and Miller, J. Comparing and combining software defect detection techniques: a replicated empirical study. SIGSOFT Softw. Eng. Notes 22, 6 (Nov. 1997), pp. 262-277.

[12] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, Y. Shaham-Gafni, Model traceability. IBM SYSTEMS JOURNAL. Vol. 45, No. 3, 2006.

[13] B. Ramesh and M. Jarke, ''Toward Reference Models for Requirements Traceability,'' IEEE Transactions on Software Engineering 27, No. 1, 58–93 (January 2001).

[14] O. C. Z. Gotel and A. C. W. Finkelstein, ''An Analysis of the Requirements Traceability Problem,'' Proceedings of the First International Conference on Requirements Engineering, Utrecht, The Netherlands (1994), pp. 94–101.

[15] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merio. Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering, 28(10), October 2002.

[16] Microsoft Solutions Framework http://www.microsoft.com/msf

[17] Kroll, Per; MacIsaac, Bruce; Agility and Discipline Made Easy - Practices from OpenUP and RUP, Addison-Wesley Professional, 2006, 448 p.

[18] Kruchten, Philip: The Rational Unified Process: An Introduction. Addison-Wesley, 2003.

[19] Conway's Life Game, http://en.wikipedia.org/wiki/Conway's_Game_of_Life