Well-defined Software Process as Vehicle to Understand Effectiveness of Formal Methods

Shigeru Kusakabe, Yoichi Omori, and Keijiro Araki

Grad. School of Information Science and Electrical Engineering, Kyushu University
744, Motooka, Nishi-ku, Fukuoka-shi, 819-0395, Japan

Abstract. In addition to software development techniques such as formal methods, software process is important for effectively developing quality software. In the literature, we have well-defined process templates which provide mechanisms for measuring, controlling, managing, and improving the way we develop software. Well-defined process templates can serve as a vehicle to integrate advanced software engineering techniques, such as formal methods, into actual software development. We expect that students who have mastered such a well-defined process template can effectively use formal methods and understand the effectiveness of formal methods. In this paper, we report on a trial case of team software process in which students tried using a model-oriented formal specification language in a lightweight way for their project. We generated a hypothesis that students with some process capabilities understand the effectiveness of rigorously writing specifications in a formal specification language while students with less capabilities do not.

1 Introduction

Formal methods are useful to effectively develop quality software. However, students who have been taught formal methods seem to seldom use them in their actual software development process. We need methods to convince students of the effectiveness of formal methods in their actual software development.

In addition to formal methods, software process is important in order to effectively develop quality software. For example, there is a claim "effective processes provide a vehicle for introducing and using new technology in a way that best meets the objectives" in a document of process improvement framework [1]. We have been working on the effective methodology to introduce formal methods into actual software development. One approach is a combination of a formal method and a well-defined customizable process template which has mechanisms of measuring and analyzing process data as well as mechanisms of process improvement. Such a well-defined process template is useful in gaining perspective for the impact of formal methods and evaluating the actual effectiveness by using process data. We expect that students who have mastered such a well-defined process template can effectively use formal methods and understand the effectiveness of formal methods in their actual software development process.

Our current focus is the combination of a model-oriented formal method we can use in a lightweight way and a well-defined customizable process template supported by a process improvement framework [2, 3]. An example of model oriented formal methods

Copyright © 2015 for this paper by its authors. Copying permitted for private and academic purposes.

is VDM and an example of well-defined customizable personal process templates is PSP (Personal Software Process) [4] which is a base process template for a team level process TSP (Team Software Process) [5, 6]. Many of recent developers are familiar with semi-formal modeling notations such as diagrams in UML. We expect that model-oriented formal methods are a good candidate with which developers customize their process as a part of their process improvement without filling a wide gap.

In this paper, we report lessons learned from a trial case of team software process in which students tried using a model-oriented formal specification language in a lightweight way for their project. As we have not collected strong evidence as in controlled experiments, we discuss our trial descriptively here. In Section 2, we explain our approach of integrating formal methods into well-defined software processes. In Section 3, we explain PSP and our trial of the combination of VDM and PSP. In Section 4, we explain TSP and our trial of the combination of VDM and the introductory TSP, before concluding in Section 5.

2 Formal Methods and Well-defined Process

2.1 Well-defined Software Process Framework

We use TSP and PSP, developed and managed by CMU (Carnegie Mellon University) SEI (Software Engineering Institute), as our baseline process templates. These are widely known as scalable, measurable and customizable processes [7]. They have a strong relation to the organization level process improvement model, CMMI (Capability and Maturity Model Integration). PSP and TSP are streamlined in the CMMI framework [8]. CMMI builds organizational capability, TSP improves team performance, and PSP builds individual skill and discipline. The CMMI framework has Measurement and Analysis (MA) process area in its set of process areas, so that we can measure and analyze the impact of the introduction of new technology such as formal methods when using the process templates related to CMMI. We evaluate the impact of formal methods on development process according to the process data collected through the measurement and analysis mechanism in the process improvement framework.

2.2 Lightweight Formal Methods

There are various ways of using formal methods in design flows. In this paper, we use a light weight approach in which the rigor level of the design flow is Level 3 or 4 according to the seven levels of increasing rigor of design flows proposed in [9]:

Level 1: Conventional design flow, with informal specifications.
Level 2: Conventional design flow, with semi-formal specifications.
Level 3: Formal design flow, with formal specifications and without tool support. This is basically a conventional design flow in which formal specifications replace informal/semi-formal ones. Most of quality control and quality assurance remain achieved using conventional quality steps.

Copyright © 2015 for this paper by its authors. Copying permitted for private and academic purposes.

Level 4: Formal design flow, with formal specifications and lightweight checking tools. The specifications are written in formal languages equipped with syntax and static semantics checkers, which can detect shallow mistakes (e.g., syntax errors, undeclared identifiers, type inconsistencies, etc.).
Level 5: Formal design flow, with formal specifications and bug hunting tools.
Level 6: Formal design flow, with formal specifications and proof tools.
Level 7: Formal design flow, with formal specifications, proofs, and proof checking tools.

There are no direct relationships between this rigor level and the levels in CMMI, capability level and maturity level. However, we can use formal methods for the important process elements such work-products and practices in process improvement.

We basically recommend a lightweight approach when starting to use formal methods. In a lightweight approach, we develop a formal specification and then work-products int the next level from the specification informally. Lightweight approaches to formal methods seem cost effective and likely to be adopted in a wider range of actual development, while the specific level of rigor depends on the goal of the project. In lightweight formal methods, we do not rely on very rigorous means, such as theorem proofs. Instead, we use adequately less rigorous means, such as writing specification in a formal specification, evaluating pre/post conditions and testing the specification, to increase our confidence in the specification.

3 Formal Methods and Well-defined Personal Process

3.1 PSP: Personal Software Process

We briefly introduce PSP without going into the details, as the details of PSP can be found in the literature [4]. PSP has different levels of process practice and can be used in education of process development and improvement as well as in actual projects. In a standard training course of PSP, development of process is divided into three levels. PSP0 features on measurement practices, PSP1 on estimation practices, and PSP2 on quality practices as shown in Fig. 1.

As shown in Fig. 2, PSP0 and PSP1 consist of the following phases: planning, detailed design, code, compile, test, and postmortem. PSP2 has extra two review phases: planning, detailed design, design review, code, code review, compile, test, and postmortem. In PSP2, four specification templates are provided: operational, functional, state and logic specification templates. Developers can improve quality of their products by using these specification templates and reviewing them with a review checklist in PSP2. Fig. 3 shows scripts of Detailed Design and Design Review phases in PSP2. Developers can measure and analyze the effectiveness of these templates and activities related to these templates through the measurement and analysis mechanism in PSP.

Once developers have established their development process as in PSP2, we expect they are ready to introduce formal methods in their process improvement. PSP was developed based on the CMM (CMMI's predecessor), and designed to be CMM level 5. PSP has a process improvement mechanism as in CMM. In the continuous representation of CMMI, the set of process areas for level 2 includes MA, Measurement

Copyright © 2015 for this paper by its authors. Copying permitted for private and academic purposes.

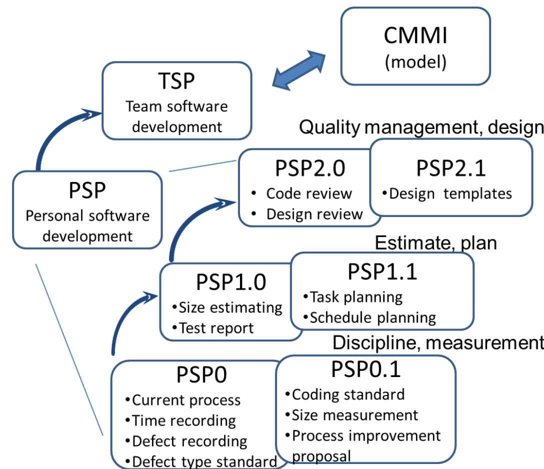

Fig. 1. Development steps in PSP

and Analysis process area. For example, we can focus on the defect types which are frequently injected and expensive to fix according to the process data once we have established such practices in PSP that reflects MA in CMMI. In addition, we can customize the baseline process template to include the work-products and activities with formal methods effective in preventing injection of defects of the specific defect types [2].

3.2 PSP with VDM

The four specification templates in PSP can be replaced with those written in formal specification languages. One instance is an approach with a model-oriented formal method VDM, one of the longest-established formal methods for the development of computer-based systems. By using VDM, we can enhance and improve the work-products and activities in Detailed Design and Design Review phases. We can use the standard language the VDM Specification Language (VDM-SL) and dialects such as VDM++, which supports the modeling of object-oriented and concurrent systems. There exist textbooks corresponding to these languages such as [10, 11]. Support for VDM includes commercial and academic tools for analyzing VDM specifications, including support for testing and code generation [12, 13]. In VDM, we can write explicit-style executable specifications as well as implicit-style non-executable specifications. For explicit-style executable specifications, we can use the interpreter to evaluate pre/post conditions in the specifications and test the specifications.

In our trial using VDM in a personal process [2], the developer changed the level of rigor in the design flow from Level 2 to Level 3 or 4. He used VDM++ instead of UML, and used VDMTools for syntax and type checking, and testing key parts of the specification potentially related to specific concerns. He spent more effort in design and

Copyright © 2015 for this paper by its authors. Copying permitted for private and academic purposes.

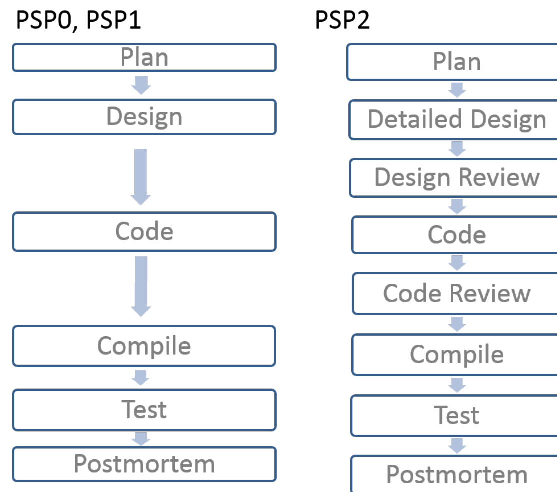

Fig. 2. Phases and process flow in PSP

design review, and less on coding and testing after introducing VDM into his process. By introducing formal methods like VDM, we can expect a reduction of undesirable behaviors like designing in coding, a reduction of defects and consequent reduction of time spent in testing, while we need extra effort for work-products and activities related to VDM. He successfully reduced the number of defects he had focused on without degrading his productivity. He had the impression that without a defined process template like PSP he could not have made a process improvement plan with formal methods.

4 Formal Methods and Well-defined Team Process

4.1 TSP: Team Software Process

After considering the lessons learned from our trial on PSP mentioned above, we continue to extend our trial to a team-level software development process, TSPi (Team Software Process introduction) [6]. While TSP is a scalable team process usable for a large scale industry project, TSPi is a subset of TSP and suitable for a small team in an academic setting. In TSPi, we strategically develop the target product throughout the multiple development cycles as shown in Fig. 4. We start from the development of the basic requirements and restart the next development cycle for the updated requirements after finishing one development cycle in an incremental manner.

Each cycle is divided into eight phases, launch/re-launch, strategy development, planning, requirements, design, implementation, testing, and postmortem phase. TSPi is a defined and customizable team process, and has scripts that cover issues such as what kind of work to be done in each phase and how to collect process data. As TSPi is

Copyright © 2015 for this paper by its authors. Copying permitted for private and academic purposes.

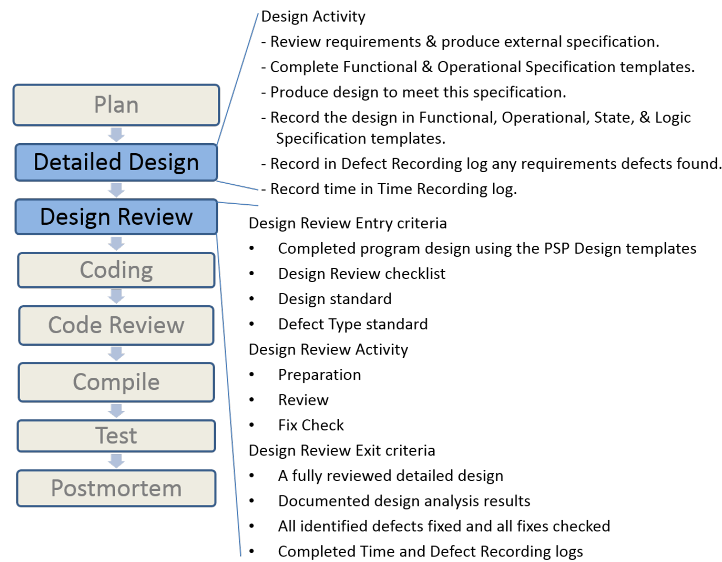

Fig. 3. Scripts of Detailed Design and Design Review Phases in PSP2

a defined process that can be tailored, we can integrate formal methods with TSPi, and evaluate the effectiveness using its process data.

4.2 TSPi with VDM

In this case, we focus on problems related to design activities among team members. In TSPi, one or two members develop a high-level design, and each member is assigned his/her task based on the high-level design. Each member develops the detailed design for his/her task before proceeding to the coding task in the coding phase. In our trial, we compare the defect data when using the normal TPSi design flow with those when we enhance the design flow with VDM in order to observe the effectiveness of using VDM.

In this case, the team used VDM as follows.

1. The team members follow the standard script and templates for the design phase in TSPi.
2. The team members complete reviews and inspections.
3. The team members collect and the record the defect data for the design phase.
4. The team members redo the design using VDM.
5. The team members use VDMTools for syntax and type checking of their design. In addition, they use the tool for animating the selected part of the design.
6. The team members collect and the record the defect data additionally found by using VDM.

Copyright © 2015 for this paper by its authors. Copying permitted for private and academic purposes.

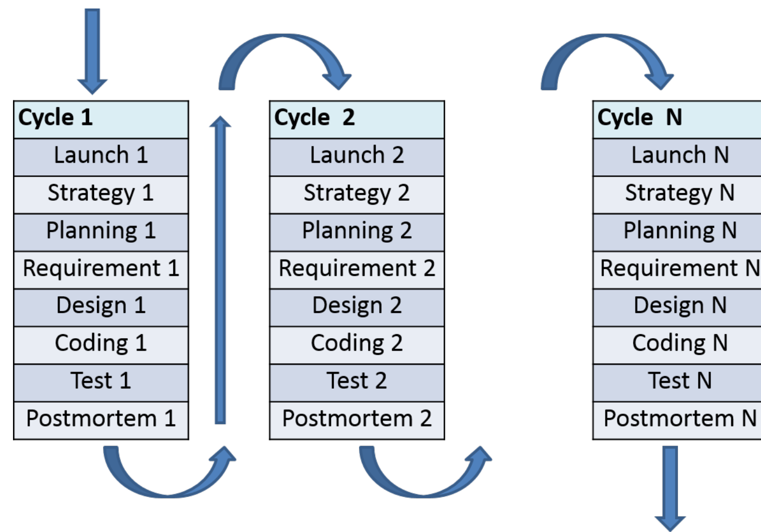

Fig. 4. TSP/TSPi process structure

Among the defects, we picked up the defects related to interface, data type, and functionality in the discussion below. This means we exclude the minor defects like typing errors. During the standard TSPi design flows, they found 2 defects in the high-level design and 37 defects in the detailed design. By using VDM, they additionally found 23 defects. These were the defects of the high-level design itself and the defects of the detailed design caused due to the flaw in the high-level design. While the members might not be practiced in reviewing and inspecting, they missed defects in the design of the normal design flow using non-formal and semi-formal notations.

Ideally, members of TSPi are expected to be practiced with their personal software process capabilities. Students are required to follow the monitoring practices like specific practices in Measurement and Analysis process area in CMMI-DEV throughout the PSP course. They also need to submit reports that require capabilities like those in Causal Analysis and Resolution process area. In our TSPi trial, members were not equally practiced with their personal software process capabilities. A few of them were practiced and the rest of them were not. According to our impression from interviews with the members, more practiced team members seemed to understand the effectiveness of formal methods more concretely while less practiced member seemed to feel more vaguely. We generated a hypothesis that students with higher process capabilities understand effectiveness of rigorously writing specifications in a formal specification language while students with lower capabilities do not.

Copyright © 2015 for this paper by its authors. Copying permitted for private and academic purposes.

5 Concluding Remarks

We have been working on the effective methodology of introducing formal methods into the actual software development process. We reported our trial case of software process in which students tried using a model-oriented formal method in a lightweight way for their project. We only have very limited data so far. However, we generated a hypothesis from our trial case that students with some process capabilities can better understand the effectiveness of rigorously writing specifications using formal method while students with less capabilities do not. We will continue and extend our effort to collect evidence to support our hypothesis.

Acknowledgment

This work was partly supported by KAKENHI, Grant-in-Aid for Scientific Research(S) 24220001.

References

1. CMMI Product Team, CMMI for Development, Version 1.3, CMU/SEI-2010-TR-033, 2010.
2. S. Kusakabe, Y. Omori and K. Araki. A Combination of a Formal Method and PSP for Improving Software Process, Proc. of TSP Symposium 2012, CMU/SEI-2012-SR-015, 2012.
3. S. Kusakabe, H. Lin, Y. Omori, and K. Araki, Generating Supportive Hypotheses in Introducing Formal Methods using a Software Processes Improvement Model, Proc. of the ICSE 2014 Workshops - 2nd FME Workshop on Formal Methods in Software Engineering (FormaliSE 2014), pp.24-30, 2014
4. W. S.Humphrey. PSP: A Self-improvement Process For Software Engineers, Addison-Wesley, 2005.
5. Team Software Process, http://www.sei.cmu.edu/tsp/
6. W. S. Humphrey. Introduction to the Team Software Process, Addison-Wesley, 2000.
7. C. Jones. Software Engineering Best Practices, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2010.
8. J. McHale, and D. Wall. Mapping TSP to CMMI, Carnegie Mellon University Software Engineering Institute Technical Report CMU/SEI-2004-TR-014, 2005.
9. Hubert Garavel, Susanne Graf, Formal Methods for Safe and Secure Computers Systems, BSI Study 875, 2003
10. J. Fitzgerald and P. G. Larsen. Modelling Systems: Practical Tools and Techniques in Software Development. Cambridge University Press, 1998.
11. P. G. Larsen, P. Mukherjee, N. Plat, M. Verhoef, and J. Fitzgerald. Validated Designs For Object-oriented Systems. Springer Verlag, 1998.
12. VDMTools, http://vdmtools.jp/en/
13. Overture Tool, http://overturetool.org/

Copyright © 2015 for this paper by its authors. Copying permitted for private and academic purposes.