VDM over PSP: A Pilot Course for VDM Beginners to Confirm its Suitability for Their Development

Hisayuki Suzumori Haruhiko Kaiya Kenji Kaijiri
Dept. of Computer Science, Shinshu University
Wakasato 4-17-1, Nagano 380-8553, Japan
chisa@cs.shinshu-u.ac.jp kaiya@cs.shinshu-u.ac.jp kaijiri@cs.shinshu-u.ac.jp

Abstract

Although formal methods seem to be useful, these is no clear way for beginners to know whether the methods are suit for them and for their problem domain, before using the methods in practice. We propose a method to confirm the suitability of a formal method. The method is realized as a pilot course based on the PSP. A course mentioned in this paper is designed for a typical formal method, VDM. Our course also helps beginners of VDM to learn VDM gradually and naturally. During the course, they can confirm its suitability as follows; First, they practice several exercises for software development, while techniques of VDM are introduced gradually. Second, process data and product data of software development are recorded in each exercise. Third, by evaluating these data by several metrics, they can confirm the suitability of VDM for their work.

Key Words

Vienna Development Method, Formal Method, Personal Software Process, Software Process Improvement, Software Metrics, Software Engineering Education.

1. Introduction

Any software development methods can not work well, unless a method is suit for users and their problem domain. Before using the method in practice, such users normally decide whether they use it, by reading its guide book and/or its review articles, by attending its tutorial, or only by believing its reputation. This is one of the reason why users usually feel awkward in using unfamiliar methods such as formal methods. If each user can understand the advantages and disadvantages of such methods by themselves, formal methods will be used more and more.

Here we briefly discuss the advantages and disadvantages of a formal method. We suppose a typical formal method *The Vienna Development Method*, VDM. By using VDM, software developers can systematically confirm the correctness of software behavior thanks to its mathematical notation for software design. By using CASE tool offered by IFAD[1], VDM users can save time and effort. Of course, there are many disadvantages in VDM. For example, it is difficult to get feedback from users and testers even though animation and simulation tools are available. Graphical notations such as use cases and/or state charts are more adequate to get such feedback[10]. In addition, developers feel to do the same thing, because specification and design documents written in VDM are often similar to source codes written in programing languages.

We discuss the reasons why formal methods are rarely used, even though they have great advantages. Of course, one of the biggest reason is their disadvantages mentioned above. However, the disadvantages are only generalization and sometimes they are only reputation. Each user does not normally confirm the advantages by himself. So another reason why the methods is rarely used is that there is no way for each user to confirm whether a method is suit for them and their problem domain.

In this paper, we propose a method to confirm whether VDM is suit for users by themselves. We realize the method as a course of software development using VDM. The outputs of the course are evaluated by metrics we proposed. Although the user is required to pay only a little efforts for confirming suitability, he can decide by themselves to use VDM.

Because users should collect the data of their software process and product for deriving the metrics, we design our course based on The Personal Software Process^{SM}, the PSP [8]. Because we focus on the changed parts of processes and products according to gradual introduction of VDM, we as-

[1] In this paper, we used VDM and VDMTools developed and licensed by IFAD - www.ifad.dk.

Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03)
0730-3157/03 $ 17.00 © 2003 IEEE

sume that users who try to do our course have already mastered techniques in the PSP, such as recording and scheduling tasks. We call our course as *VDM over PSP* because of the backgrounds above. We use the word *'the PSP'* in order to distinguish the original PSP from our extension.

The rest of this paper is organized as follows. In Section2, we briefly introduce underlying techniques, the PSP and VDM. Section3 is the core part of this paper. In the section, we show the structure of our course and metrics for confirming VDM's suitability in detail. In Section4, we illustrate how to use our course by using an example. Finally, we summarize current results and show future's direction.

2. Underlying Techniques for VDM over PSP

In this section, we briefly introduce techniques for building our course, underlying techniques for VDM over PSP.

2.1. The PSP

The goal of software engineers is to develop reliable software within given cost and schedule[9]. The PSP proposes scheduled planning and quality management method in personal level.

The PSP includes the following techniques[13]: measuring product and process (In particular, they report time log, defect log and software size), recording the data into forms along with the process script, and analyzing the data. During the introductory programming course using the PSP [8] , PSP learners learn the above techniques. In this course, they experience improving software process by gradually introducing methods. Software process improvement is achieved by evaluating the data with respect to several metrics. The evaluation of the PSP also contributes for students to know the benefits of particular methods[6, 12].

2.2. Formal Methods

Formal methods are used to design software by using mathematics. Specifications in formal methods can be checked formally using some mathematical techniques and tools. In general, formal methods contribute the improvement of software development as follows [4, 10].

- Because the specifications can be described exactly, there is little misunderstanding between writers and readers in reading such specifications.
- By analyzing the specification formally, defects can be found in early phase, and most defects can be found at the end.

In addition, formal methods are not used frequently by the following preconceived notion[10].

- Because specifications written in formal methods are often similar to source codes written in programing languages, developers feel to repeat the same thing.
- Because mathematics are used in formal methods, developers feel that formal methods are difficult.

2.2.1 VDM and VDMTools

In this paper, we use a typical formal method, VDM. VDM was designed by IBM Vienna laboratory in 1973[4]. There are two specification languages for VDM, VDM-SL and VDM-SL++, and we use VDM-SL in this paper. CASE tools are also available such as VDMTools developed by IFAD [3]. We only use VDM-SL Toolbox(in the following Toolbox means VDM-SL Toolbox) in VDMTools. We only use the following functions in Toolbox, because VDM over PSP is for beginners of VDM.

Syntax check :To check syntactic legality in VDM-SL.

Type check :To check type compatibility statically.

Interpreter and Debugger :Toolbox has an interpreter and source level debugger for VDM-SL. It is possible to check invariants and preconditions dynamically.

We decided that functions in VDM and Toolbox are gradually introduced in VDM over PSP, so that you can feel your improvement which is achieved by the functions.

3. VDM over PSP

In this section, we introduce the structure of our course and metrics for confirming VDM's suitability.

3.1. Course Structure

In the same way as the PSP, our course has also four levels, from VDM over PSP0 to VDM over PSP3. The baseline of our process, VDM over PSP0, is the same as level 2.1 of the PSP. Along with the progress of the course, techniques of VDM are gradually introduced.

3.1.1 Where is VDM embedded?

Because VDM over PSP use PSP2.1 as the baseline, we briefly review the structure in the PSP2.1. The PSP2.1 consists of two kinds of objectives, one is for cost estimation and schedule planning and another is for quality management. For quality management, review and semi-formal design specification techniques are mainly used. Design

Table 1. VDM over PSP 3 process

	name of the phasis	name of phases
start	design phasis	planning
		design (Modified)
		design review
		VDM-SL syntax review
⇓		**Syntax check (with Tool)**
		Type check (with Tool)
		Validation (with Tool)
	implementation phasis	code
		code review
end		compile
		test
		postmortem

templates are used for writing design specifications, and review guidelines and checklists are used during its review phase. Because VDM can be used as an alternative of design template, it can be used for quality management instead of design templates in the PSP2.1. Therefore, we select the PSP2.1 as the baseline of VDM over PSP.

3.1.2 Process Levels in VDM over PSP

We have already mentioned that PSP2.1 is the baseline process of VDM over PSP. Here we introduce each level of VDM over PSP in detail. Along with the progress of the level, techniques in VDM are gradually introduced. Exercises in our course are categorized into the following levels;

VDM over PSP0 : The baseline personal process. This is the same as the PSP2.1.

VDM over PSP1 : Functional specifications in VDM-SL are introduced. In addition to VDM over PSP0, VDM over PSP user uses VDM-SL description to specify data types, pre-conditions, implicit function descriptions and invariants of each data type.

VDM over PSP2 : Logical specifications in VDM-SL are introduced. In addition to VDM over PSP1, the user specify the internal specification of each function.

VDM over PSP3 : Using Toolbox to validate specification written in VDM-SL. In this level, the user uses the process as shown in Table1, the quality management metrics that are redefined as describe in section 3.1.5, and procedures as shown in Table2. We will describe the detail of this level in the next section.

3.1.3 An Overview of the Process Script of VDM over PSP3

In Table1, we show a process script of VDM over PSP3. We categorize that the first seven phases are design phasis and

Table 2. Validation procedures using the design template and ToolBox

phase	the design template	step
function check	functional specification template	1. VDM over PSP user uses interpreter only, doesn't use optional function in interpreter. 2. The user uses interpreter, selects option for dynamic possible well-formedness in interpreter. 3. The user uses interpreter, selects option for dynamic possible well-formedness and dynamic definite well-formedness in interpreter.
use case check	operational scenario template	1. The user uses interpreter only, doesn't use optional function in interpreter. 2. The user uses interpreter , selects option for dynamic possible well-formedness in interpreter. 3. The user uses interpreter, selects option for dynamic possible well-formedness and dynamic definite well-formedness in interpreter.

Table 3. Category of VDM template

	internal	external
static	Explicit function description (VDM)	Data type description (VDM) Invariant relation of data types (VDM)
dynamic	State specification template (the PSP)	Implicit function description (VDM) Pre-condition description (VDM) Operational scenario template (the PSP)

the rest are implementation phasis. Scripts of other levels in VDM over PSP are subsets of this script. Here we explain how to make VDM to cooperate with the PSP. Because the quality management in the PSP has two aspects: one is the defect preventive strategy and another is the defect elimination strategy, we discuss this cooperation in each aspect respectively.

For the defect preventive strategy in VDM over PSP, we also use review guidelines and checklists in the same way as the PSP. We categorize VDM-SL descriptions into design template in the PSP as shown in Table3. This table shows the category of complete design representation for design templates in the PSP[8]. VDM over PSP users use new design templates for VDM instead of design templates of the PSP. However, we use two design templates of the PSP, state specification template and operational scenario

template, because we can not describe such kinds of specifications naturally in VDM-SL.

For the defect elimination strategy in VDM over PSP, we append four additional phases into the script in Table1. In addition, we use design templates at the validation phase, as the guideline to generate test case. The design phase, which is also the part of the PSP, are slightly modified in VDM over PSP. The details of the additional and modified phases are as follows;

Design : VDM over PSP user describes design specification in VDM-SL.

VDM-SL syntax review : the user checks VDM-SL specifications to find syntax defects. Although VDM-SL specifications can be executed, defects should be checked by the review as well as in defects of the code.

Syntax check : the user executes the syntax checker in Toolbox.

Type check : the user executes the type checker in Toolbox.

Validation : the user validates the specification by executing it with the interpreter. In this phase, poof techniques are not used, because our course is for beginners of VDM.

In Table2, we show validation procedures using design templates and ToolBox. Columns of the table show the name of the phase, design template witch is used to generate test data, and the ordering of the step achieved in each phase. In these procedures, VDM over PSP users check the validity of each function first, and the validity of each scenario next. By using external and invariant specification of each function and reasonable test data, they can check the former kind of validity. By using each scenario written in operational scenario template, They can check the latter kind of validity.

3.1.4 New Defect category

Because VDM-SL specification is executable, syntax defects of VDM-SL should be recorded in the same way as code defects in the PSP. These defects are only injected in design phasis. After the code phase, these defects are not injected. We append new defect categories for defects of VDM-SL specification because we formally distinguish code defects from those of VDM-SL specification. Syntax defects of VDM-SL contribute to know design quality.

In the PSP, defect recording log consists of the following members for each defect: the phase where defect is removed, the phase where defect is injected, category of defect from defect type standard in the PSP, and removal time. In VDM over PSP, we use the same form as well.

In the PSP, defects are categorize into 10 types. The first four types represent code syntax defects, and the rest types represent design defects [9]. We add four additional types for VDM-SL syntax defects in the same way as code syntax defects.

3.1.5 Redefining The Quality Management Metrics

We redefine the Cost-Of-Quality (COQ) metrics. COQ are used for evaluating the process quality in the PSP. COQ indicates economics of defect removal. There are two kinds of COQ in the PSP: failure cost and appraisal cost. The defect removal phases are classified into either failure cost or appraisal cost.

We classify the cost of the validation phase in the failure cost because the validation phase is almost the same as test. We classify the cost of syntax check and type check phases in the failure cost because the phases are almost the same as compile phase. We classify the cost of VDM-SL syntax review phase in the appraisal cost because the VDM-SL syntax review phase is review phase.

We don't need to redefine the metrics for quality management. The other metrics are the same as those in the PSP.

In VDM over PSP3, VDM over PSP users follow the script in Table1 to develop a program. They describe design specification to design templates in Table3. They use review guidelines and check lists in each review phase. They use procedures in Table2 at the validation phase. They evaluate our process and product using the quality management metrics.

3.2. Metrics for Confirming VDM's Suitability and Unsuitability

Here we propose the new metrics to confirm VDM's suitability from the product and process data.

3.2.1 Metrics for Confirming VDM's Suitability

We define three metrics for confirming VDM's suitability as follows; The metric focuses on design defect because we suppose that design quality influences design defects. We define the metric for confirming VDM's suitability based on quality management metrics of the PSP.

- Ratio of Phases where design defects are removed(DDR)
  $\mathrm{DDR}(phase_i) = \frac{\mathrm{design\_defects\_removed\_in\_}phase_i}{\mathrm{all\_design\_defects}} \times 100$
  Objective: VDM over PSP users can remove almost all design defects up to the validation phase. After the validation phase, They can leave no design defect.

- Ratio of phases where design defects are injected(DDI)
  $\mathrm{DDI}(phase_i) = \frac{\text{defects\_injected\_in\_}phase_i}{\text{all\_design\_defects}} \times 100$
  Objective: They can find bad final design decision in early phase.

- Design defect removal leverage(DDRL)
  $\mathrm{DDRL}(phase_i) = \frac{\text{Defects/Hour}(phase_i)}{\text{Defects/Hour(UnitTest)}}$
  Objective: They can remove design defect easily.

Note that $phase_i$ means i-th phases in Table1.

3.2.2 Metrics for Confirming VDM's Unsuitability

We define two metrics for VDM's unsuitability as follow:

- Productivity
  $\mathrm{Productivity}(exercise_i) = \frac{\text{LOC}}{\text{total\_development\_hour}}$
  Objective: The decline of productivity in VDM over PSP's development.

- Number of Design Defect per KLOC(NDDK)
  $\mathrm{NDDK}(exercise_i) = \frac{\text{all\_design\_defects}}{\text{KLOC}}$
  Objective: The rise of design defects per KLOC in the user's development.

Note that $exercise_i$ means i-th exercise in VDM over PSP course.

3.2.3 How to use the Metrics

We explain how to use our metrics to indicate VDM's suitability and unsuitability. In the same way as the PSP, several number of exercises are imposed to an user who tries to do our course. Along with the progress of the course, VDM over PSP user can observe the changes of metrics. the user can identify the effects and/or suitability of VDM from such changes. We observe only the change of each metrics along with the progress of VDM over PSP course. A way of how to evaluate data measured with the metrics is shown in Table 4.

4. How to use Our Course: An Example

In this section, we illustrate how to use our course and how to confirm VDM's suitability using an example. Note that data here does not come from real experiences of our course, and merely a fiction based on an experience of the PSP. We have not applied our course to any persons yet.

The situation for this example is as follows.

> A software engineer engages in a mission critical project. Fortunately, there are enough budget and time for the project. Because he is well-experienced engineer, he has already used quality management techniques such as review and semi-formal design. For improving the quality of his process furthermore, he tries to introduce VDM.

Assume VDM over PSP is used under such situation. Nine exercises for software development are prepared for his VDM over PSP. As shown in Figure 1, 2, 3 and 4, exercise 1 and 2 are solved by VDM over PSP0, 3 and 4 by VDM over PSP1, 5 and 6 by VDM over PSP2 and 7, 8 and 9 are solved by VDM over PSP3. Problems in the exercises belong to the same domain as the problem of his project. Now, he tries to confirm VDM's suitability for him by using our metrics.

4.1. Confirming Suitability

As mentioned in Section3.2.1, we have proposed three metrics for confirming VDM's suitability; Ratio of Phases where Design Defects are Removed (DDR), Ratio of Phases where Design Defects are Injected (DDI) and Design Defect Removal Leverage(DDRL). The engineer using our course tries to plot each metrics in each exercise. In Figure 1, 2 and 3, X axis shows the progress of exercises.

Figure1 shows the changes of the ratio DDR during the course. Up to the level of VDM over PSP2, there are no significant changes of the ratio. That is to say, about 90% of defects are removed before test phase. At the level of VDM over PSP3, where validation using VDM-SL tool is introduced, the ratio is significantly changed. That is to say, about 90% of defects are removed before code review phase. From the results, the engineer feels that VDM is useful for him when tool support is available, but it is not so useful only writing formal specifications.

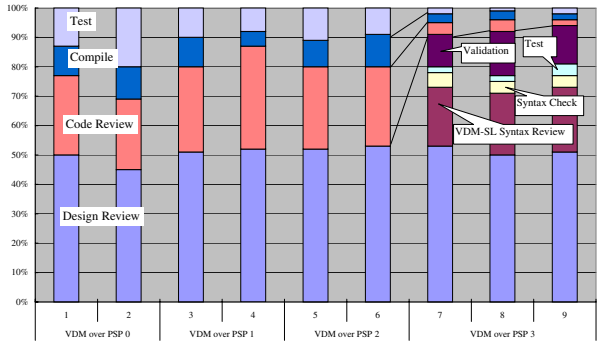

Figure 1. Ratio of Phases where Design Defects are Removed (DDR)

Figure2 shows the changes of the ratio DDI during the course. As the course is progressed, the ratio of design defects injected in coding phase is gradually decreased. Because design defects injected in coding phase implies that design decision is not fully discussed in design phase, he finished the lager number of design decision in earlier phase. At the level of VDM over PSP3, there are few de-

Table 4. What does the change of each metric indicate and What can be concluded?

Metrics	Increase	Decrease
DDR of design phasis	VDM over PSP user removed more design defect in design phasis and the user removed design defect earlier. So, VDM is useful for design defect elimination.	The user removed insufficient number of design defects in the phasis and activities in the phasis did not contribute to eliminate design defects. So, the user may introduce other methods instead of VDM when the user removes design defects in design phasis.
DDI of design phase	The user increased ratio of the design phase where design defects are injected and the user can finished the larger number of design decision than in the previous exercise. So, VDM contribute to find bad finish design decision in earlier phase.	The user tended to overlook design issues during design phase. So, VDM put design issues out of the user's sight and the user should stop using VDM immediately.
DDRL of design phasis	The user got DDRL of a phase of design phasis more than the other phase and the user could remove design defects more efficiently. So, VDM techniques introduced in the phase will contribute to improve efficiency of design defect removal.	The efficiency of design defect removal became worse. So, the user may introduce other methods instead of VDM with respect to the efficiency of design defect removal when the user removes design defect in the phase.
Productivity	The user's productivity was improved more than the previous exercise. So, VDM is useful for cost saving.	The user productivity became smaller than the previous exercise. So, the user should examine whether the user have enough budget and time for using VDM.
NDDK	If the user's ability of defect detection was not changed during the course, bad design decision was increased. So, the user may introduce other methods instead of VDM for design.	If the user's ability of defect detection was not changed during the course, bad design decision was decreased. So, VDM is useful to improve the design quality.

sign defects injected in coding phase. From the results, the engineer feels that VDM is useful for him, and only writing formal specifications will helps to decrease design defects injected in coding phase.

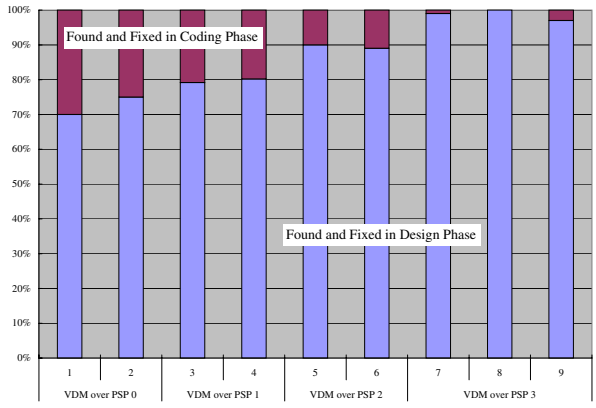

Figure 2. Ratio of Phases where Design Defects are Injected (DDI)

Figure3 shows the changes of DDRL during the course. In the same case in Figure1, there are no significant changes of the ratio up to the level of VDM over PSP2. In all exercises, a leverage by design review is extremely stronger than leverages by other phases. However, leverages by VDM-SL syntax review and validation are not so bad, and leverages by code review are improved at the level of VDM over

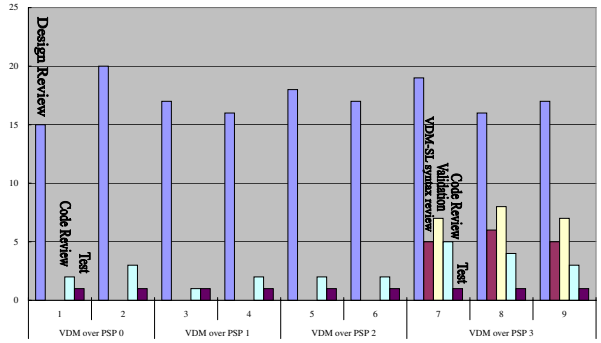

Figure 3. Design Defect Removal Leverage (DDRL)

PSP3. From these results, the engineer feels that VDM is useful for him when tool support is available, but it is not so useful only writing formal specifications.

4.2. Confirming Unsuitability

As mentioned in Section3.2.2, we have also proposed two metrics for confirming VDM's unsuitability: productivity and the number of design defects per KLOC. The engineer using our course tries to plot each metrics in each exercise in Figure4 and 5. X axis also shows the progress of exercises.

As shown in Figure4, his productivity is declined as the

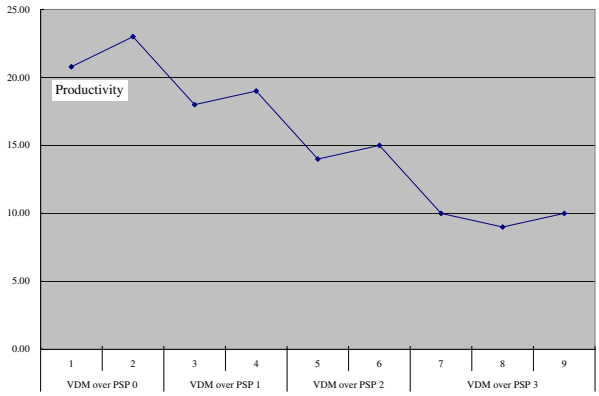

Figure 4. Productivity

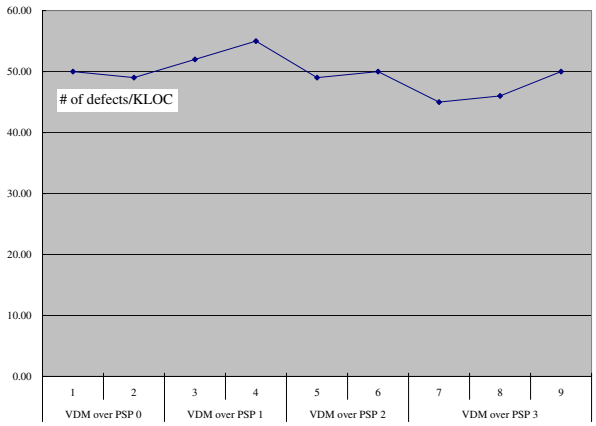

Figure 5. Number of Design Defects per KLOC

course is progressed. By approximating the line graph in Figure4 into a straight line, the coefficient of its decline is about `-1.8`. From the result, the engineer feels that VDM is harmful for efficiency. However, he do not mind this fact because his project has enough budget and time!

As shown in Figure5, the number of defects per KLOC is not changed as the course is progressed. By approximating the line graph in Figure4 into a straight line, the coefficient of its decline is about `-0.5`. So we may say that the number is slightly decreased. From the result, the engineer feels that VDM gives no effects to the amount of design defects.

4.3. Summary of his Decision

As a result, the engineer tries to use VDM with tool support in the project he engaged. The reasons are as follows. First, he feels VDM and its tool slightly contribute to remove design defects. Second, he feels VDM contributes to decrease the injection of design defects after design phase. Third, he do not need to mind productivity in the project.

5. Conclusions and Future Works

In this paper, we propose a pilot course, namely VDM over PSP. By using this course, a beginner of VDM can confirm whether VDM is suit for him and for his problem domain systematically. Although, a few costs are required for the course before using VDM into practice, each software engineer can decide by himself to use VDM. As mentioned in the first section, we assume that people who try to do our course have already mastered techniques in the PSP. This seems big fetter of our course. However, we believe that our course can motivate engineers to learn such disciplines in the PSP, because they can be free from public reputation of methods and techniques by our course.

There are two kinds of design quality, one is the quality of the design representation and another is the quality of the design contents. The former quality is more important than we have imagined, because poor representation usually causes poor implementation[8]. Unfortunately, metrics for the former kind of quality are not proposed in both the PSP and VDM over PSP. We want to propose such metrics in the future. In IEEE standard 830[1], we can find quality characteristics for the requirements specification such as modifiability and traceability. Concepts of such characteristics will helps us to build metrics for design representation.

As shown in Table2, we propose the process script for validating VDM-SL specification using Toolbox. However, the script is not complete because we should prepare test cases for achieving the process but there is no clear way for preparing them. We want to append techniques for generating test cases to our course.

As implied in the fist section in this paper, we regard that each personal software process using VDM is mainly characterized by both the person who engages the process and by the problem domain he engaged. By performing our course by himself, the former kind of characteristics can be specified, but the latter kind of characteristics can not. By specifying the problem domain in our course, we should also design the contents of exercises. For example, exercises in the PSP[8] seem to be designed for a problem domain where numerical and statistical calculation is important. We should offer another kind of exercise series when an engineer engages in another kind of problem domain.

Another big problem of our course is the cost for achieving the course. From the experiences of the PSP course, it is very very hard for any engineers to complete this kind of course, because it requires big time and efforts. If you can engage a project with enough budget and time, we strongly recommend to do our course for confirming VDM's suitability. If you can't, this is the ordinary case, records of

other engineers will help you to decide whether you use VDM. We want to build a database for such records in the future.

We will apply VDM over PSP to a lecture for undergraduate students. We have already found several reports of lectures using the PSP[6, 7, 12, 11, 14], and most lectures use the subset of the PSP. The reason is that there is not enough time and effort in such lectures. Because there will not be enough time and effort too in our course, such reports provide us with useful information to simplify VDM over PSP.

In this paper, we only focus on a typical formal method, VDM. In the future, we want to generalize our course about design methods. In the next step, we want to propose a course for confirming model checking techniques, because it can be suitable for problem domain where VDM is not. Formal taxonomy of design methods such as method base [2] will be useful for such generalization.

Acknowledgments

The authors would like to thank IFAD for offering free academic site licenses for VDMTools[5]. This work is supported by the foundation for C&C Promotion, Japan.

References

[1] IEEE Recommended Practice for Software Requirements Specifications, 1998. IEEE Std. 830-1998.

[2] S. Brinkkemper, M. Saeki, and F. Harmsen. A Method Engineering Language for the Description of Systems Development Methods. In *CAiSE'2001 Proceedings*, pages 473–476, 2001. LNCS 2068.

[3] I. Company. *VDM-SL Toolbox User Manual*, 2001.

[4] J. Fitzgerald and P. G. Larsen. *Modelling Systems, Practical Tools and Techniques in Software Development*. Cambridge University Press, 1998. VDL-SL, Toolbox Lite.

[5] Free Academic Site Licenses for VDMTools. `http://www.ifad.dk/Products/VDMTools/free_asl.htm`.

[6] R. F. Grove. Using the personal software process to motivate good programming practies. *ITiCSE'98 Dublin,Ireland*, pages 98–101, 1998.

[7] T. Hilburn and M. Towhidnejad. Doing quality work: the role of software process definition in the computer science curriculum. *in Proceeding of 28th SIGCSE Technical Symposium on Computer Science Education*, pages 277–281, 1997.

[8] W. S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, 1995. The Complete PSP^{SM} Book.

[9] W. S. Humphrey. *Introduction to the Personal Software Process*. Addison-Wesley, 1997.

[10] J. Jacky. *The way of Z – practical programming with formal methods*. Cambridge University Press, 1997.

[11] K. Lisack. The personal software process in the classroom: Student reactions(an experience report). *in Proceeding of 13th Conference on Software Engineering Education & Training*, 2000.

[12] J. I. Maletic, A. Howald, and A. Marcus. Incorporating psp into a traditional software engineering cource: An experience report. *CSEET'01*, 2001.

[13] L. Prechelt and B. Unger. An experiment measureing the effects of personal software process(psp) training. *IEEE TRANSACTION ON SOFTWARE ENGINEERING*, 27(5):465–472, May 2000.

[14] M. Towhidnejad and T. Hilburn. Incorporating the personal software process(psp) across the undergraduate curriculum. *in Proceeding of ASEE/IEEE Frontiers in Education Conference*, November 1997.