VDM++ as a Basis of Scalable Agile Formal Software Development

Hiroshi Mochio[1] and Keijiro Araki[2]

[1]Chikushi Jogakuen University, 2-12-1 Ishizaka, Dazaifu-shi, 818-0192 Fukuoka, Japan
[1]`mochio@chikushi-u.ac.jp`
[2]Kyushu University, 744 Motooka Nishi-ku, Fukuoka-shi, 819-0395 Fukuoka, Japan
[2]`araki@csce.kyushu-u.ac.jp`

Abstract. This paper presents the use of VDM++ formal specification language as a basis of scalable agile formal (SAF) software development method, which is an agile method for mission-critical or large-scale software development. Combination of agile method, of which usefulness has been recently recognized, and VDM++, a formal method, enables describing system architecture, verifying specification and generating source code all in an agile and formal way. The system architecture and the verification are indispensable for SAF software development.

Keywords: agile method, formal method, scalable development

1 Background

It has been about ten years since the Agile Manifesto was declared. [1] During this period, agile software development method has been more and more adopted in real industry field and realistic results have been obtained. The significance of the agile development method is already undoubted today. Agile development principles, such as iteration based on lean requirements, good communication among the stakeholders and regularly responding to changes, lead to satisfaction of developers and customers. Consequently, it brings about high productivity.

The method of lean requirements enables developers to catch an outline of the iteration that they are working on. Therefore, the developers can easily understand the whole requirements and always keep their motivations high. Good communication enables developers to keep products of high quality because they can recognize a gap between the status of the products and the requirements. Iteration of a short term development enables developers to cope with changes of the market and the customer flexibly.

Contrarily, there have been several comments about proper problems of the agile development method. Lack of ability to develop mission-critical software and lack of scalability are typical ones.

About the lack of ability for mission-critical software, Kent Beck, who promotes eXtreme Programming (XP), one of the most commonly used agile development methods, notes that “more than XP” may be needed in safety-critical systems. It means that in such situations additional process measures such as documented traceability, formal design review by outside experts and the like, may also be required. [2]

In XP, reliability of software is built up by means of test and review. Since XP is a test-driven development method, at first, tests must be described before design, and then verification by tests is performed at every stage in the iteration. Additionally, pair-programming, so to say, a kind of real-time peer review, is the basic coding style of XP. Therefore, software produced through XP always has high quality, but more reliable means of verification, for example, verification by proof, is needed for safety-critical domains.

Documentation in agile development is a little peculiar because of the principle that agile method is based on iteration for lean requirements. In agile iteration, developers should select the functions to be realized from the ordered list of requirements by themselves, and implement them. In such situation “working software over comprehensive documentation (Agile Manifesto)” is the most important, and so, no rigorous document about requirements or specifications exists. This is because there is a tacit understanding that a developing team can get the iteration into perspective. However, when high safety is required, specification about safety must be shared through the whole development process. In other words some kind of means to describe the specification strictly is needed.

On the other hand it has been pointed out in relation to scalability of agile software development that in case of a large-scale project, developers cannot get the whole development process into perspective. Namely agile development method is suitable only for not so large-scale a project. Kent Beck says as follows.

For the first iteration, pick a set of simple basic stories that you expect will force you to create the whole architecture. Then narrow your horizon and implement the stories in the simplest way that can possibly work. At the end of the exercise you will have your architecture. [3]

This means that if a system is of modest scope such that a small number of stories and an iteration or two can lay out a reasonable architectural baseline, then this approach may be very effective, and architecture can emerge quite nicely in this model. [2] Conversely, if the system is not of modest scope, it may be difficult to lay out an architectural baseline. It is the point of criticism about the scalability of the agile method.

Formal method is the generic name for methods to describe and verify a system based on mathematics, which were originally studied in Europe in the 1970’s. Formal methods are usually applied to the upper stage of development so as to exclude ambiguity or errors of specification by rigorous description and verification. These days the application cases to industry field are increasing. As a result, it is found that formal methods are effective in reinforcing safety of a mission-critical system or reducing regression processes in software development.

There are two major usages of formal methods. One is formal specification description and another is formal verification. For the former, formal specification languages, such as Z, B, VDM and OBJ are used. The typical method of the latter is model checking, and many kinds of model checking systems, such as SMV, SPIN and LTSA, are available. Usually the right formal method is applied to the right place. It is unusual to apply only one formal method to all over the development process rigidly.

Sometimes formal methods are criticized because of cost increase with its introduction or developers' antipathy against it.

The reason of the cost increase is that at least one new process for formal description or verification must be added to the upper stage of development. Besides, in the initial introduction process of formal methods, cost of training developers is necessary. Nevertheless some report says that these costs are offset by the decrease of regression processes in lower stage of development, which leads to reduction of the total cost.

The antipathy of developers is more critical than the cost increase. Indeed many of developers hesitate to make use of formal methods in spite of admitting the effectiveness of them. In such cases people are likely to regard the difficulty of formal methods as the reason for the hesitation. However, if analyzed in detail, it becomes clear that there is another factor of the hesitation. The true reason developers often reject formal methods is that formal methods are usually introduced into traditional predictable development process such as "the waterfall." Since the whole detailed specification of a system must be defined in the early stage of such process, developers are forced to take great pains in upper stage. The developers do not resist the difficulty of formal methods, but resist the difficulty to specify such a system as consists of not predictable factors all at once. [4] If the scope of the target system is modest and the perspective on it is easy to be detected, actually the difficulty of formal methods does not matter.

Agile method and formal method were originated in respective contexts, and have been developed separately. They are often taken as conflicting methodologies. But, the same as the two concepts, "agile" or "formal", are not opposed to each other, the two methodologies are mutually compatible. Appropriate combination of them rather results in more efficient and higher quality development method, because each can get rid of the other's problem. [5], [10]

In this paper, VDM++ is introduced as the core of a SAF development method, which is applicable to mission-critical or large-scale software. First, requirements for SAF method and those for the core formal method are showed. Second, after a brief overview of VDM++, it is presented that VDM++ can work as the core formal method of SAF development. Third, realizability of SAF method is discussed referring to a case of VDM++ application from industry and a software development environment with VDM++.

2 Scalable Agile Formal (SAF) Software Development

The requirements for SAF method are roughly divided into those to deal with mission-critical systems and those to deal with large-scale systems.

The former requirements are satisfied by formal specification description and verification, which are primary functions of formal methods. Rigorous description based on mathematics and detailed verification makes it possible to achieve the required high-level reliability or safety.

The latter are satisfied mainly by these three means. [2]

- Intentional Architecture
- Lean Requirements at Scale
- Managing Highly Distributed Teams

Leffingwell explains them as follows.

An intentional architecture typically has two key characteristics: (1) it is component-based, each component of which can be developed as independently as possible and yet conform to a set of purposeful, systematically defined interfaces; (2) it aligns with the team's core competencies, physical locations, and distribution. Agile teams should organize around components, each of which can be defined/built/tested by the team that is accountable for delivering it. Moreover, because of the existence of a set of interfaces that define a component's behavior, teams can be isolated from changes in the rest of the system. Sufficient architecture must be established prior to substantive development. In the first few iterations, a primary version of architecture is built and tested. The architecture should be implemented, not just modeled. This process is called "architectural runway." [2]

Requirements that define performance, reliability, and scalability of a system must be understood by all teams who contribute to the solution. To this end, requirements, which are naturally lean, should have three main elements: a vision, a roadmap, and just-in-time elaboration. The vision carries the common objectives for the teams and serves as a container for key nonfunctional requirements that must be imposed on the system as a whole. The roadmap illustrates how that vision is to be implemented over time in accordance with a prioritized backlog. Just-in-time requirements elaboration defers specific requirements expression until the "last responsible moment," eliminating the waste and scrap of overly detailed. [2]

At scale, all agile is distributed development. Tooling is also different for distributed teams. Larger teams require relatively more tooling, and at enterprise level, a more systematic approach is required. Enterprise-level communication environment needs shared, program-wide visibility into priorities, real-time status and signaling, and dependencies. Teams must have access to networks and Internet-enabled tools that enable shared repositories for agile project management, shared requirements, source code management, a common integrated build environment, change management, and automated testing. [2]

In addition to the above-mentioned requirements, the core formal method of SAF development must meet the requirements of "ordinary" agile method. Since one of the most important characteristics of agile methods is test-driven development based on the principle of "working software over comprehensive documentation," high-quality working software is demanded at every the end of each iteration. Therefore, ability to describe tests, framework for automated tests, function to animate specification, and an automated code generation tool are indispensable to the core formal method environment.

Taking all mentioned above into consideration, requirements for the core formal method for SAF software development are the following.

1. Rigorous description and verification
2. Test-driven development
3. Object-oriented description of architecture
4. Animation of specification
5. Just-in-time requirements elaboration
6. Automated code generation
7. Internet-enabled tool for communication

3 VDM++ as a Basis of SAF Software Development

VDM++ is an object-oriented extension of the formal specification description language for Vienna Development Method (VDM), which is a formal methodology originally developed at the IBM Vienna Laboratory in the 1970's. VDM++ is a product of the Aphrodite project in EU and its original, VDM-SL, was internationally standardized as ISO/IEC13817-1 in 1996. It can express many kinds of abstract data types based on mathematical equipment, such as propositional or predicative logic, set, mapping, and so on. Hence VDM++ can describe objects in a variety of abstraction levels, from an abstract model like system architecture to a concrete component. It rigorously defines functional behaviors of a system with explicit description of preconditions, postconditions, and invariants. Both implicit and explicit styles are available for definition of a function. Implicit style definition, which defines "what to do" without any description of concrete processing, declares relation between the input and the output as functional specification. It is usually used to describe highly abstract models. On the other hand, explicit ones, which define "how to do" the output from the input in terms of algorithm, can run on an interpreter, and so enables prototyping and verification by animation.

It is worth mentioning that VDM++, a formal specification language, resembles to ordinary programming languages in description style. The following is a part of an example in Fitzgerald 2005. [6] It describes the specification of an operation to return the schedule of experts who are called up by the alarm system of a chemical plant.

```
class Plant
…
public ExpertIsOnDuty: Expert ==> set of Period
ExpertIsOnDuty(ex) ==
  return {p | p in set dom schedule &
        ex in set schedule(p) }
end Plant
```

In Java, it would look something like:

```
import java.util.*;

class Plant {

  Map schedule;

  Set ExpertIsOnDuty(Integer ex) {
    TreeSet resset = new TreeSet();
    Set keys = schedule.keySet();
    Iterator iterator = keys.iterator();

    while(iterator.hasNext()) {
      Object p = iterator.next();
      if (((Set)
        schedule.get(p)).contains(ex))
        resset.add(p);
    }

    return resset;
  }
}
```

The VDM++ description looks like that of an ordinary programming language and is familiar to most programmers. Meanwhile it is, as is characteristic of formal specification languages, more abstract than the Java description, where it captured the essentials of the object simply.

Here the VDM++ adaptability to the SAF requirements mentioned above is considered step by step.

Rigorous description and verification: VDM++ can rigorously define behavior of a system with preconditions, postconditions, and invariants in function specification. Additionally it can verify the specification by satisfiability check for implicit definition or integrity check for explicit one. [6]

Test-driven development: VDMUnit is a test framework for VDM++, which is a transplant from JUnit developed for Java by Kent Beck and Eric Gamma. The figure 1 is an overview of VDMUnit framework. [6]

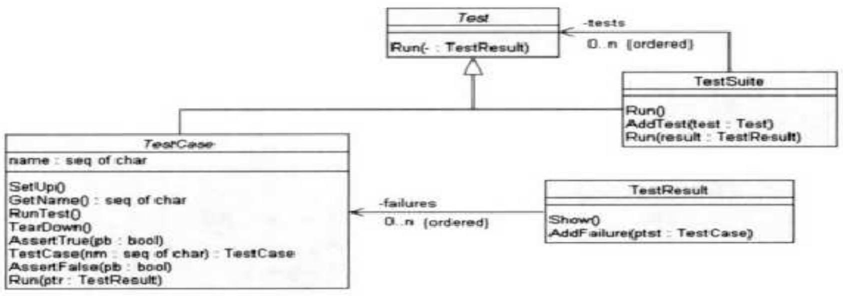

Figure 1. An overview of VDMUnit test framework

Object-oriented description of architecture: VDM++ can describe system architecture that defines components and their interfaces of a system in variety of description styles, from the abstract one for concept models to the concrete one for detailed specification.

Animation of specification: One of the most important principles of agile development is constant and close communication with customers. Customers should usually check the products during iteration, so that tasks of the iteration will comply with their stories, which are, so to say, requirements of agile style. Thus moving objects should be demonstrated for the customers to grasp the development state easily. In VDM++ development environment, specification described in explicit style can be animated with interpreter.

Just-in-time requirements elaboration: VDM++ has an abstract data type, called "token", with which VDM++ describes, without concrete definition, such objects as is not necessary to the modeling for the moment. Moreover, when function specification is described in implicit style, no more detailed definition is needed until concrete specification is demanded.

Automated code generation: There are two major kinds of description environments for VDM++, VDMTools of CSK and Overture of the Overture Project. Each of them has function to generate Java/C++ source code from VDM++ specification.

Internet-enabled tool for communication: Tooling, such as above-mentioned VDMTools or Overture, must be expanded and enriched.

4 Prospects

While a large-scale development case with VDM++ and support tooling for VDM++ are presented, realizability of SAF software development is considered.

FeliCa Networks Company developed the firmware of mobile FeliCaIC chip from 2004 until 2006. They used VDM++ to describe the external specification of components, which had about 100,000 lines, while the C++ source code of the firmware had about 160,000. [7, 8] The whole development was based on traditional waterfall process, but some kinds of formal methods were partly introduced to improve the process. According to Kurita, there were four major purposes in applying formal methods: (1) rigorous specification description all through the development, (2) achievement of high quality in upper stage of development through high-precision description and tests, (3) thorough testing based on specification, and (4) activation of communication within the development team and with customers. As a result they had corrected many faults in the upper stage, and therefore no bug has been detected since the product was released.

Though the project followed the traditional waterfall development process, it implies possibility of VDM++ architecture description in SAF development, because the external specification of the components was described with VDM++. Besides, the fact that the tests was based on the formal specification with VDM++ and the fact that VDM++, a formal method, contributed to activation of communication, both of them are hints of realizability of SAF development with VDM++.

The Overture project is an open-source project to develop new generation tooling for VDM. The mission of the Overture project is twofold: (1) to provide an industrial-strength tool that supports the use of precise abstract models in any VDM dialect for software development. (2) to foster an environment that allows researchers and other interested parties to experiment with modifications and extensions to the tool and different VDM dialects. [9]

Overture is an integrated development environment (IDE) built on top of the Eclipse platform. The core element of Overture, called VDMJ, consists of parser, abstract syntax tree, type checker, interpreter with test coverage function, and integrity examiner. Overture is the integration of VDMJ, editor, file navigator, debugger, and formatting tool, which can be evolved by expansion plug-ins.

Today, large-scale development is almost inevitably done in distributed circumstances. Therefore network-enabled powerful IDE is indispensable to SAF development. Overture is one of the likeliest candidates for such an IDE.

5 Concluding Remarks

With VDM++ and its supporting tool, system architecture can be described, components can be specified and verified, and the source code can be generated, consistently. Thus SAF software development, an agile development method for mission-critical or large-scale systems, is possible with VDM++.

Future works are as follows.

Overture IDE needs to be expanded for scalable agile development. The first thing to do is to equip online video chat and whiteboard-like system as circumstances for real-time communication among all concerned. Secondly test-driven function should be improved so that tests can be generated and run automatically. Additionally, specification animation with enriched interface is desirable because such function is useful especially for communication between a developer and a customer. Then reinforcement of prototyping and code generating function is needed, for working software is the most important thing in agile development process. Finally a team management tool is necessary for making development teams correspond to the components defined as elements of system architecture.

References

1. Beck, K., Grenning, J., et al.: Manifesto for Agile Software Development. http://agilemanifesto.org/ (2001)
2. Leffingwell, D.: Scaling Software Agility: Best Practices for Large Enterprises. Pearson Education, Inc. (2007)
3. Beck, K.: Extreme Programming Explained: Embrace Change. Boston, MA: Addison-Wesley. (2000)
4. Fowler, M.: The New Methodolgy. http://www.martinfowler.com/articles/newMethodology.html (overhauled in 2005)
5. Black, S., Boca, P., Bowen, J., Gorman, J., and Hinchey, M.: Formal Versus Agile: Survival of the Fittest? In: Computer, September 2009. IEEE Computer Society, pp. 37-45. (2009)
6. Fitzgerald, J.S., Larsen, P. G., Mukherjee, P., Plat, N., and Verhoef, M.: Validated Designs for Object-Oriented Systems. Springer-Verlag London. (2005)
7. Kurita, T., Chiba, M. and Nakatsugawa, Y.: Application of a Formal Specification Language in the Development of the "Mobile FeliCa" IC Chip Firmware for Embedding in Mobile Phone. In: FM 2008: Formal Methods, Springer, pp. 425-429. (2008)
8. Kurita, T. and Nakatsugawa, Y.: The Application of VDM to the Industrial Development of Firmware for a Smart Card IC Chip. International Journal of Software and Informatics, Vol. 3, No. 2-3, pp. 343-355. (2009)
9. Overture Community: Overture: Formal Modelling in VDM. http://www.overturetool.org/
10. Larsen, P.G., Fitzgerald, J.S., Wolff, S.: Are Formal Methods Ready for Agility? A Reality Check. In: Grüner, S., Rumpe, B. (eds.) FM+AM 2010, 2nd Intl Workshop on Formal Methods and Agile Methods. Lecture Notes in Informatics, vol. 179, Gesellschaft für Informatik, 2010, pp. 13-25. (2010)