Int J Software Informatics, Vol.3, No.2-3, June/September 2009, pp. 343–355 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
©2009 by Institute of Software, Chinese Academy of Sciences. All rights reserved. Tel: +86-10-62661040

The Application of VDM to the Industrial Development of Firmware for a Smart Card IC Chip*

Taro Kurita and Yasumasa Nakatsugawa

(Sony Corporation, Tokyo, Japan)

Abstract We have applied the formal specification language in the development of the firmware of the smart card IC chip for embedding in mobile phone. We report on an industrial application of formal methods to the development of a complex system, namely the firmware for the "Mobile FeliCa" smart card IC chip. The use of formal techniques, specifically the Vienna Development Method (VDM), was aimed at raising the quality of system specifications by reducing ambiguity and improving communications between engineers. Development data gathered during the life cycle confirm the effectiveness of a lightweight formal method in contributing to the quality of the deliverables in early development stages. No software specification problems have, to date, been reported since first release (over 100 million mobile phones have the chip embedded).

Key words: Formal Methods; Formal Specification; Formal Modeling; VDM; VDM++

Kurita T, Nakatsugawa Y. The application of VDM to the industrial development of firmware for a smart card IC chip. *Int J Software Informatics*, **2009, 3(2-3): 343–355. http://www.ijsi.org/1673-7288/3/343.htm**

1 Introduction

This paper reports on an application of the Vienna Development Method (VDM)[1] to the Mobile FeliCa IC chip development project by Sony Corporation[2]. The focus of this work was to be able to produce more precise specifications than conventional methods allow and, hopefully, as a result to increase the quality of the firmware developed. In order to assess the efficacy of VDM for this project, several development metrics were gathered.

"FeliCa" is a contactless IC card technology widely used in Japan, developed and promoted by Sony Corporation[3]. This technology is utilized in the Mobile FeliCa IC chip which is embedded in a mobile phone. Mobile phones embedded with the FeliCa IC chip are known as "Osaifu Keitai" (literally "mobile wallet") by NTT DOCOMO, Inc.[4], and as of today over 100 million mobile phones have the chip embedded. The chip enables phones to be used as electronic cash, train tickets, personal identification, door keys etc.

The Mobile FeliCa system (Fig. 1) is comprised of mobile phones containing the FeliCa IC chip, FeliCa servers connected to the mobile telecommunications network

*Corresponding author: Taro Kurita, Email: taro.kurita@jp.sony.com

Manuscript received 2009-02-14; revised 2009-08-16; accepted 2009-08-18.

and FeliCa reader/writers. The Mobile FeliCa IC chip firmware provides: a secure file system and communications protocol, which are the bases of the FeliCa technology; and firewall functions that enable the multiple services in the Mobile FeliCa IC chip such as electronic cash and train tickets. The IC chip firmware and the hardware itself were developed from scratch for the second generation Mobile FeliCa system. Multiple semiconductor manufacturers developed the hardware in collaboration with Sony Corporation while Sony Corporation was fully responsible for all stages of firmware development from defining requirements through to deployment and maintenance.

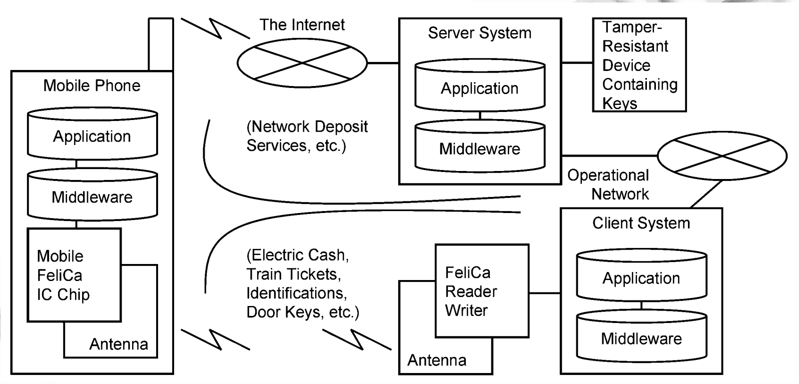


Figure 1. Mobile feliCa system

The high volume means that it would be extremely expensive to perform a product recall once the chips embedded in the mobile phones. It was therefore essential to ensure the extremely high quality of the software in order to avoid the serious social infrastructure problems that could affect stakeholders if defects were present.

Before beginning this development, we analyzed difficulties that had arisen in previous development projects. The key issues we identified were ambiguity in specifications and communication difficulties between engineers. Previous projects had defined specifications using a combination of natural language and UML and, in general, they contained ambiguities. As a consequence, the implementation, testing, deployment and maintenance were not carried out in a correct manner. The ambiguities also caused a communication overhead between specifiers, developers and testers because it was necessary to identify who was responsible for the clarification as well the necessary clarification itself. This had contributed to insufficient software quality and too inefficient development. Formal specification techniques claim to offer a reduction in specification ambiguity and provide a sound basis for communication between engineers, and have a record of industry use[5]. It was therefore felt to be worth investigating the use of a formal specification language to gain the improvements sought.

In the remainder of this paper, we describe the goals we had before starting this application of VDM in Section 2. Section 3 gives a short introduction to the VDM technology selected for this project. In Section 4 the approach taken in to structuring the development teams and processes is presented. This is followed in Section 5 with a presentation and evaluation of the results collected from the project.

Finally, concluding remarks are provided in Section 6 and future issues considered for the next project using VDM are presented in Section 7.

2 Goals of Applying the VDM Technology

After reviewing the lessons of past projects and considering the character of the Mobile FeliCa IC chip development project, we decided to focus on making improvements to the design phase of the software development process as well as creating mutual understanding between engineers. The specific goals were as follows:

Creating precise specifications: This goal was to eliminate ambiguities, omissions and duplication in specifications. The desire was to obtain a precisely defined specification so that implementation, deployment and maintenance could be systematically and efficiently developed.

Enhancing the quality of deliverables at the design phase: The intention here was to write the firmware functional specifications in VDM in parallel with producing a natural language specification for developers downstream of the specifiers using formal techniques. It was hoped that this would improve the quality of the specification documents because the production of the VDM model would detect errors that might do undetected in the natural language formulations. In doing this we hoped to be able to check the satisfiability of the requirements.

Improve development processes: The goal here was to improve existing processes for specification development, firmware implementation and testing, but to do so in an evolutionary manner by incorporating construction and validation of VDM models, and exploiting them in subsequent implementation and testing activities.

Testing thoroughly at different levels: This goal was to take advantage of a precise VDM models to exploit testing throughout the phases of a project life cycle and not only at the end of a project.

Improving communications between engineers: This goal was to take advantage of the formal specifications as a common basis for communication between different teams of engineers during specification development, implementation, testing, deployment and maintenance.

3 Formal Specification Languages and Tools

For the Mobile FeliCa IC chip We decided to use the formal specification language VDM++[6] and VDMTools[7,8], since they support describing and validating large-scale models. VDM++ is a multi-purpose formal specification language that is primarily an object-oriented extension of VDM-SL[9] which is a formal specification language standardized under the International Organization for Standardization[10,11].

VDMTools is an integrated tool that supports VDM model analysis via syntax and type checking, proof obligation generation and testing. Implicit (contractual pre-/post-condition) specification and executable explicit specification styles are supported. The tools possess features for management of specifications and round trip engineering to and from UML class diagrams. An interpreter provides debugging support and subsequent test coverage analysis for VDM models written in the language's large executable subset. VDMTools also contains automatic code generators for C++ or Java. However, these were not designed to produce code for embedded systems for secure smart IC card platforms and so this feature was not used.

4 Approach

The artifacts produced in the overall development process are shown in Fig. 2. Developed products are shown with solid lines and tools used in development are shown with dashed lines. Although the hardware interfaces are all the same, IC chip hardware instruction sets and compilers are different. We therefore needed to develop software separately for each target.

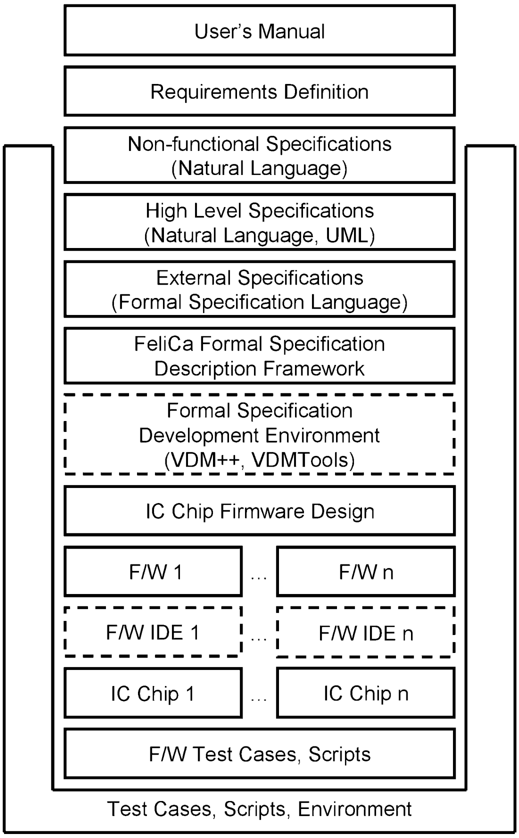

User's Manual
Requirements Definition
Non-functional Specifications (Natural Language)
High Level Specifications (Natural Language, UML)
External Specifications (Formal Specification Language)
FeliCa Formal Specification Description Framework
Formal Specification Development Environment (VDM++, VDMTools)
IC Chip Firmware Design
F/W 1 … F/W n
F/W IDE 1 … F/W IDE n
IC Chip 1 … IC Chip n
F/W Test Cases, Scripts
Test Cases, Scripts, Environment

Figure 2. Project artifacts

We developed and tested specifications of external behavior using the formal specification language. All functional specifications were written in VDM++. The main components or functions covered by the formal specifications are as follows:

- The FeliCa file system specification that defines the basic data structure.

- Wireless and wired interfaces of the Mobile FeliCa smart card.
- The framework for describing and testing specifications that are based on the basic data structure and interfaces.
- Command specifications which are the basis of the FeliCa technology.
- Security specifications combined with file system and command specifications.

4.1 Description and testing of specifications

We designed a development process based on the "formal method based transformation model"[12]. We discussed and wrote requirements with stakeholders and wrote high level specifications in natural language with various diagrams based on UML notation, such as state transition diagrams and sequence diagrams.

From these requirements, we designed and implemented a common formal specification description framework in VDM++. This framework served as a common basis for the formal specification of the firmware. Because of the wide variety of specifications and specification styles supported by a flexible formal specification language, such a framework was desirable in order to ensure consistency across a large scale project, and hence achieve a viable, implementable design. Defining basic terminology, data format, templates and a testing framework creates an easy-to-read specification, so that engineers involved in design are able to concentrate on describing and testing the specification.

Having developed a framework, the next step was to describe specifications of the FeliCa file system, interfaces, commands and security. This was done in a non-operational implicit style. An implicit specification describes the characteristics of objects to be expressed such as data format and state, invariants, preconditions satisfying the data state before an operation and postconditions satisfying the data state afterward. VDMTools provide syntax and type checking and document generator features of implicit specifications.

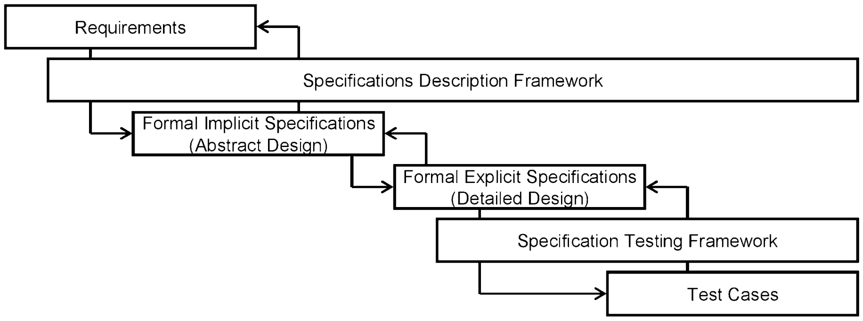

Requirements
Specifications Description Framework
Formal Implicit Specifications (Abstract Design)
Formal Explicit Specifications (Detailed Design)
Specification Testing Framework
Test Cases

Figure 3. Specification development process

Finally, we developed a framework for writing and testing executable explicit specifications, and wrote explicit specifications and test cases. Explicit specifications is able to test and debug much like software. At the time of execution, it was possible to dynamically check invariants, preconditions and postconditions. Non-functional specifications such as performance and reliability were written in natural language separately from the formal specifications.

The process of specification development is shown in Fig. 3.

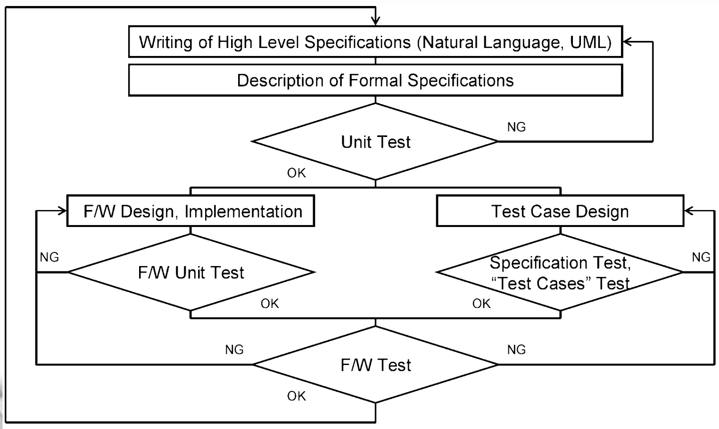
4.2 Teams, training and development process

The project engineers worked in three teams which varied in size: a specification team of 5-20 members, a firmware implementation team of 15-20 and a testing team of 25-35. No members had prior knowledge or experience with formal methods at the time of project launch. Not all of them are well acquainted with software engineering, and there were even members who had no software engineering experience at all.

As for the skill level of the engineers, although all had learned the basics of information technology, there was a wide gap in the number of years of experience in software development. At the time of project launch, some of the members received five days of training from a formal methods specialist. Other members received in house training sessions. After the initial phase of employing a formal method, we continued to retain specialist consultants and examined other case studies of application of formal methods[6,13,14].

We used an iterative development approach. A single iteration can be described as a self contained development life cycle, starting from describing specifications to testing the firmware on a development board. One iteration consumes a few weeks, and in total 30 iterations were carried out.

The outline of the single iteration of the development process is shown in Fig. 4.



Writing of High Level Specifications (Natural Language, UML)
Description of Formal Specifications
Unit Test
NG
OK
F/W Design, Implementation
Test Case Design
NG
F/W Unit Test
Specification Test, "Test Cases" Test
NG
OK
OK
NG
F/W Test
NG
OK

Figure 4. Single iteration of the development process

The specifiers' role was to investigate and negotiate requirements with the planning division and other companies such as users of the mobile wallet platform. They would then write and modify a high-level specification using natural language and UML, write a formal specification using the specification description framework and perform unit testing on the formal specifications, as shown in the initial phases in Fig. 4.

On completion of unit testing, the specification was then handed to both implementation engineers and test engineers. The implementation engineers' role was to execute firmware design and modification according to the specification, and carry out unit testing on different hardware platforms using IC chips emulators within their development environment. The test engineers' role was to devise test cases based on the specification, develop software programs for testing, build a test environment including the latest IC chip emulator, execute testing against the specification and execute testing against the software delivered by implementation engineers.

For testing the specification, the test engineers first executed tests for consistency against the specification using test programs based on test cases. Concurrently, they examined sufficiency of test cases by conducting coverage analysis against the executable formal specification. Consequently, more test cases were added if required. Secondly, they executed test programs that were verified to be consistent to the specification against the firmware on different hardware in order to check whether behavior was as stated by the specification. Lastly, the engineers checked that the executable specification and the programs stored in ROM of different types of IC chips behaved the same. The test scheme for the overall development is shown in Fig. 5.

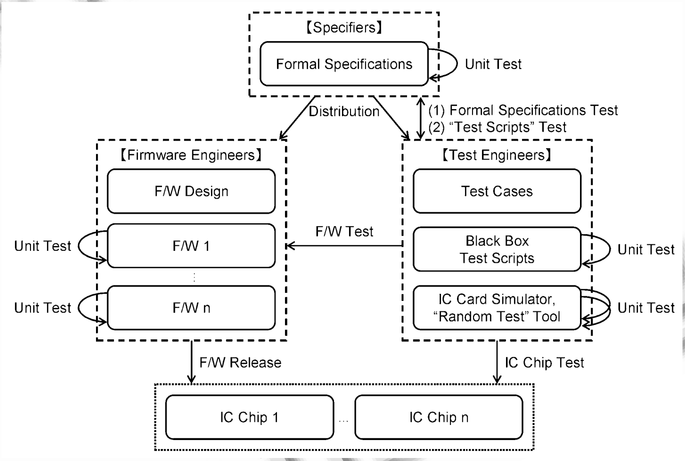


Figure 5. Test scheme

We executed mainly black box testing. From the formal specifications, test engineers designed black box functional test specifications and then implemented test scripts. Executable formal specifications, firmware on a development board and IC chips were automatically tested using the test environment and the test scripts. From the results of the tests, we were able to confirm whether the test cases and scripts were consistent with the specifications. In addition, from measuring the coverage of the executable formal specifications, we confirmed that the test cases covered all of the defined specifications.

The test engineers also designed a "Random Test" tool to continuously send random commands to the test target and check whether the test target sent back correct results as per specifications. The tool had a specification emulator feature, which created expected values based on the formal specification. The specification emulator included emulation of error handling by changing the internal status in addition to emulation in normal states. The test target was tested by comparing the test data sent back from the “Random Test” tool and that from the test target.

By carrying out about 7 000 black box test cases and 100 million random test cases, a high quality IC chip was achieved.

5 Metrics from the Project

The Mobile FeliCa IC firmware project duration was 39 months inclusive of maintenance and customer support. There were 50-60 team members in this project, and the average age was about 30 years old. We employed several chip manufacturers in order to reduce risks in manufacturing and sales. It was necessary that the firmware should behave in exactly the same way on different ICs and firmware development environments so that the compatibility was retained. C/C++ and assembler languages were used in the implementation of the firmware.

The specification outputs were as follows:

- 383 pages of a protocol manual written in natural language (user manual for other departments within the company and for outside customers)
- 677 pages of an external specification document written in formal specification language
- Our formal specifications are 140 251 lines including test cases (66 412 lines) and comments written in natural language (34 460 lines)

Using these specifications, we implemented the C/C++ code of about 164 000 lines, inclusive of headers and comments (about 100 000 lines), as firmware of two IC CPUs. At the time of writing, there have been no reported problems related to the software specifications since first release.

5.1 Tests and reviews of specifications

The results on improving specification quality by testing and reviewing the specification are shown in this section.

The line coverage rate of the formal specifications by unit testing (as shown in Fig. 4) was 82%. We were able to improve the coverage rate of unit test cases by coverage analysis supported by the VDM tools. As a result of unit testing, we were able, for example, to discover an incorrect path in postconditions. This kind of inconsistency is generally difficult to discover by review.

Formal specification errors discovered during the specification process before integration test are shown in Table 1. The use of a formal method contributed to enhancing the quality of deliverables at design phase of the development process because the formal specification can be syntax-checked, type-checked and tested. In our experience, it is very difficult to achieve the same level of quality with natural language and UML specifications that are subjected to more limited checking and inspection.

It is possible to measure a coverage rate of test cases by executing black box testing against the executable specification. The line coverage rate of the formal specifications by black box testing and visual inspection was 100%.

Table 1 Number of errors discovered before integration test

Describing Specifications	162
Executing and Unit Testing Specifications	116
Reviewing Specifications	93
Communicating with Firmware Engineers	69
Total	440

5.2 Efficiency of specification development

The average productivity for the formal specifications was about 1 900 lines of VDM per engineer per month (approximately 160 hours), including the time consumed for examination of requirements, writing a high-level specification using natural language and UML, and testing a formal specification. This is equal to the rate for implementation including firmware design and unit testing, suggesting that developer productivity is certainly not reduced, especially if the VDM is more expressive than the target code. Given that this was also a first application of formal specification by the team, it can be said that there were no particular disadvantages to using a formal specification language from scratch.

Even non-software engineering specialists were easily able to read, write and test a formal specification having received only short term training by a specialist. The difficulties sometimes attributed to formal methods were not a hindrance to applying a lightweight formal method[15,13] in our experience. Despite having no specialists in formal methods within the project, it was possible to employ a formal method with initial help from an outside specialist.

5.3 Percentages of errors in firmware implementation

The percentages of errors in firmware implementation related to specifications for the overall project are as shown in Table 2. The table presents errors in the following groupings:

- Errors attributable to the specification activity. These are broken down into:
  - Missing descriptions: specifiers failed or were unable to supply a specification for a particular case.
  - Erroneous descriptions: a specification bug.
  - Unclear description: the specification is present but was not rigorous and/or not understood by readers.
- Errors attributable to firmware design/implementation and test (although these are also in part caused by the failure of specifiers to provide specifications that are sufficiently easy to read):
  - Oversight: The specification has been described rigorously but readers read it erroneously or were unable to read it.
  - Insufficient Understanding: The specification has been described rigorously but were unable to fully understand it and the readers and specifiers did not confirm their understanding with one another.

  - Insufficient Confirmation: A failure of communication between the readers and the specifiers leading to a lack of confirmation of understanding.

- Failure of change propagation (a function of project management).
- Other causes.

Table 2 Percentages of errors in firmware implementation

Reason for Errors	Percentage
Missing description	0.2%
Erroneous description	0%
Unclear description	1.8%
Oversight	5.6%
Insufficient understanding	10.7%
Insufficient confirmation	0%
Failure of change propagation	0.2%
Others (reasons unrelated to specifications)	81.5%

It was very rare to have problems in implementation and testing caused by ambiguity in the specification. We concluded that formal methods are useful for finding errors in the early stages of development. We could say that there were very few errors left until the later stage of development, so that we could concentrate only on the design, implementation and testing based on a precise specification. The use of VDM contributed to improving both the specification and the development process and deliverables.

Based on the above results, it can be said that we have successfully described the specifications in a precise way. On the other hand, the total percentage of "oversight" errors and "insufficient understanding" errors was 16.3%. This was due to the fact that the separations between the actual specifications and the code required to execute the specifications was unclear.

Our future task is to achieve a specification that is both easily readable and executable. It is necessary to pay attention to not only executable features but also the readability of specifications. Specifications which are referred to by all project members need to be simple, so that they can be read without stress.

5.4 Comparison dost with COCOMO estimation

We estimated the cost, effort and schedule required to develop with COCOMO (COnstructive COst MOdel)[16] before project started. The estimate is calculated[17] based on the size of the application is as shown in Table 3.

The real development from specification development to firmware implementation took about 24 months and 950 person months. It is possible to conclude that our development is considerably better than the estimate from the COCOMO and application of formal method did not delay project progress. This estimation was calculated by Sony Corporation and personnel who estimated COCOMO for the Trade One project[6]. Although the development domain and the skills of engineers were different between Mobile FeliCa IC chip development project and Trade One project, in both cases more effective development and higher software quality were achieved by applying formal methods with less cost than COCOMO estimation suggested.

Table 3 COCOMO calculation parameters and results

Software Product Size	
Source Lines of Code (SLOC)	64 211 Lines
SLOC Inclusive of Headers and Comments	163 869 Lines
Software Development Mode	
	Embedded
Software Development Cost Drivers	
Product Attributes	
Required Reliability	High
Database Size	Low
Product Complexity	Very High
Computer Attributes	
Execution Time Constraint	Extra High
Main Storage Constraint	Extra High
Virtual Machine Volatility	High
Computer Turnaround Time	Low
Personnel Attributes	
Analyst Capability	Normal
Applications Experience	Low
Programmer Capability	Normal
Virtual Machine Experience	Very Low
Programming Language Experience	High
Project Attributes	
Modern Programming Practices	Normal
Use of Software Tools	Normal
Required Development Schedule	Normal
Results	
Effort	1947.06 Person Months
Schedule	28.22 Months

6 Conclusion

The application of lightweight formal techniques was viewed as highly successful in keeping our project on schedule. The formal method contributed to the quality of deliverables at the design phase of the development process. Thanks to the formal methods, there have been no problems related to the software specifications since the first release.

Considering our original goals for applying a formal method, we examine its effectiveness. It is possible to define functions by using precise specifications and possible to test executable specifications. From this perspective, high quality is achievable in the early stages of development. This assists us in achieving a correct implementation and in testing. By writing formal specifications precisely, the quality of discussion and understanding in the related domain and the specification can be improved. This makes it possible to check whether the requirements have been adequately fulfilled, and to improve the quality of not only formal specifications but also other deliverables accompanying the formal specification in the early stages of development.

Efforts later in the development cycle naturally benefited as a result of improved upstream quality. With a development process based on a formal specification, the system was developed effectively and with high quality. By making good use of a precise specification, carrying out various methods of testing from various points of view improved the quality of software, and regression testing of the specification confirmed the same quality regardless of changes.

Finally, it becomes possible to achieve effective and high quality development by project members who were able to use a common vocabulary and framework to constructively discuss shared objectives.

We conclude that formal methods are suitable for reaching high quality within the Japanese traditional philosophy of “Kaizen” — continuously improving the process step by step, working closely together with team members.

7 Future Issues

Successful validation of specifications against requirements is an important part of the development process that often involves negotiation with stakeholders who lack the skills to read formal specifications. Communication with stakeholders currently has to be done through informal specifications, so the informal models have to be produced at additional cost and introduce an extra risk of misunderstanding. We would rather write one specification once and use it everywhere. To resolve this, we would like to find ways of presenting the formal specification to stakeholders via a readily understood notation or interface. The validation techniques that we have applied so far are only as good as the test sets that are developed. Verification of logical consistency properties such as security requirements, could be addressed by more advanced verification techniques such as proof or model checking.

So far, we have applied formal specification techniques only to the firmware. To extend the benefits of our approach to the whole system (not only the functional specification but also the hardware abstraction layer) we would benefit from abstractions appropriate to the specification of embedded systems[18].

Acknowledgements

We would like to thank Professor Emeritus Dines Bjørner of Technical University of Denmark, Professor John Fitzgerald of Newcastle University, Professor Peter Gorm Larsen of Engineering College of Aarhus, Professor Keijiro Araki of Kyushu University, Shin Sahara of CSK Systems Corporation, Hiroshi Sako of Designers' Den Corporation, Miki Chiba of Sony Corporation for their great assistance in the application of the formal method.

References

[1] Fitzgerald J, Larsen PG, Verhoef M. Vienna Development Method. In Wah B (Ed), Wiley Encyclopedia of Computer Science and Engineering, John Wiley & Sons, Inc, 2008.

[2] Kurita T, Chiba M, Nakatsugawa Y. Application of a Formal Specification Language in the Development of the “Mobile FeliCa” IC Chip Firmware for Embedding in Mobile Phone. In Cuellar J, Maibaum T, Sere K. Eds, FM 2008: Formal Methods, Lecture Notes in Computer Science, Springer, 2008, 5014: 425-429.

[3] Sony Corporation. FeliCa Web Site. `http://www.sony.net/Products/felica/`

[4] NTT DOCOMO, Inc.. “Osaifu-Keitai”. `http://www.nttdocomo.co.jp/english/service/osaifu/`

[5] Woodcock J , Larsen PG, Bicarregui J, Fitzgerald J. Formal Methods: Practice and Experience,

ACM Computing Surveys, 2009.

[6] Fitzgerald J, Larsen PG, Mukherjee P, Plat N, Verhoef M. Validated Designs for Object-oriented Systems, Springer, 2005.

[7] Fitzgerald J, Larsen PG, Sahara S. VDMTools: Advances in Support for Formal Modeling in VDM. ACM Sigplan Notices, 2008, 43(2): 3-11.

[8] CSK SYSTEMS CORPORATION, VDM information web site, `http://www.vdmtools.jp/en/`

[9] Fitzgerald J, Larsen PG. Modelling Systems: Practical Tools and Techniques in Software Development, 2nd Edition, Cambridge University Press, 2009.

[10] International Organization for Standardization, Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language, ISO/IEC 13817-1, December 1996.

[11] Plat N, Larsen PG. An Overview of the ISO/VDM-SL Standard, ACM Sigplan Notices, 1992, 27(8): 76-82.

[12] Liu SY. Formal Engineering for Industrial Software Development: Using the SOFL Method, Springer, 2004.

[13] Hall A. Using Formal Methods to Develop an ATC Information System, IEEE Software, 1996, 13(2): 66-76.

[14] Jones CB. Systematic Software Development Using VDM, 2nd Edition, Prentice Hall, 1990.

[15] Hall A. Seven Myths of Formal Methods, IEEE Software, 1990, 7(5): 11-19.

[16] Boehm BW. Software Engineering Economics, Prentice Hall, 1981.

[17] University of Southern California Center for Systems and Software Engineering, COCOMO 81 Intermediate Model Implementation, `http://sunset.usc.edu/research/COCOMOII/cocomo81_pgm/cocomo81.html`

[18] Larsen PG, Fitzgerald J, Wolff S. Methods for the Development of Distributed Real-Time Embedded Systems using VDM, International Journal of Software and Informatics, 2009, 3(2-3).