A Survey on Formal Specifications in Industry

向　剣文、野田　夏子
日本電気株式会社
共通基盤ソフトウェア研究所
j-xiang@ah.jp.nec.com, n-noda@cw.jp.nec.com

本稿では、産業界における形式仕様記述のいくつかの実適用例の比較研究について述べる。これらの適用例はそれぞれが別々に既に他者により報告されたものであるが、この比較研究ではそれらの個々の事例に共通の経験や教訓を示すことを目的とする。使用された形式仕様記述言語、得られたコード品質、及びコスト(トレーニングコスト、コストシフト、費用効果性など)の点から、いくつかの知見を得ることができたので、それらについて報告する。

A Survey on Formal Specifications in Industry

Jianwen Xiang, Natsuko Noda
Common Platform Software Research Laboratories
NEC Cooperation
j-xiang@ah.jp.nec.com, n-noda@cw.jp.nec.com

A comparative study on several industrial applications with formal specifications is presented. These applications have been reported independently elsewhere before, while the comparative study is to further show some common experiences and lessons learned from these individual case studies. A number of observations are made relating to the specification languages, code quality, and cost (e.g., training cost, cost shift, and cost effectiveness) of the industrial applications with formal methods.

I. Introduction

All too often in the software industry, domain descriptions, requirements prescriptions, and software designs are written in an informal way which typically lack of clarity, focus, and confidence. These informal specifications might be incomplete, inconsistent, and/or ambiguous, and thus it can be neither rigorously analyzed for properties nor used as prototypes [1].

With mathematical and logical formalizations, formal methods (FM) can be used to find defects early in the software development life cycle. Recently, it seems that more and more researchers have noticed the importance of formal methods. To illustrate this point, we carried out a rough statistics on the literatures with the keywords "formal methods" in ACM digital library in the most recent 20 years, and the result is shown in Figure 1 (note that the result is not limited to software but may also include hardware). In spite of other possible factors (such as incomplete history data), it seems that there is a big boost in the research on formal methods from the most recent decade. Similar result also exists in IEEE Xplore but with a smaller magnitude of numbers as shown in Figure 2.

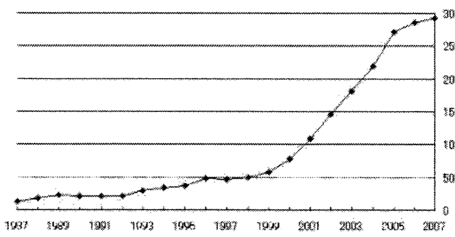

Figure 1. Literatures on formal methods in ACM Digtial Library

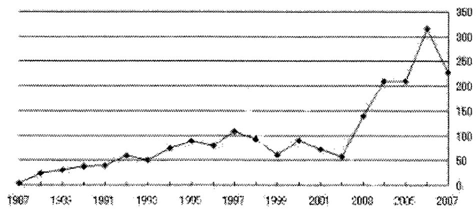

Figure 2. Literatures on formal methods in IEEE Xplore

However, the reality of industry is not such exciting as that of academia. Formal specifications so far still remain absent from most development processes, although they have gained some acceptance in areas such as safety- and security-critical systems. Lacking of *comprehensive understanding* on the experiences and lessons of formal methods is one of the key obstacles.

The aim of this article is to investigate six industrial applications of formal methods, and then to present some observations relating to the specification languages, code quality, and cost (e.g., training cost, cost shift, and cost-effectiveness) of the formal methods by a comparative study between these case studies. Individual reports of these case studies have been published elsewhere before, but a comparative study on them is hopefully to provide more comprehensive insights, which exactly constitutes the main motivation of this article.

Note that all of our observations are rather *suggestive* than conclusive due to the relatively small sample space and the "incomplete" comparisons between the samples. Our comparisons solely rely on the source data of individual case studies varying from their focuses and coverage, and thus most of the comparisons are only taken between some (two or three, but not all) of the samples which cover the particular issue of concern.

The rest of this paper is organized as follows. Section II gives a brief introduction of the industrial formal applications referred in this article. Section III presents our comparative observations in terms of the specification languages, code quality, and cost analysis of the formal methods. Finally, we carry out our concluding remarks in Section IV.

II. Summary of Formal Applications

A brief introduction to the case studies of formal applications mentioned in this article is as follows.

- The Customer Information Control System (CICS), a transition processing system developed by IBM with Z [2] in 1990 [3, 4].
- The Central Control Function (CCF) Display Information System (CDIS), a real-time air traffic control information system developed by Praxis with VDM (Vienna Development Method) [5], FSM (Finite State Machines), VVSL [1], and CCS (Calculus of Communicating Systems) in 1993 [6, 4, 7].
- The Trusted Gateway (TG) developed by BASE (British Aerospace Systems and Equipment Ltd.) using both formal (VDM) and informal (CASE technology) methods in the middle of 1990s [8, 9, 10].
- The Ship Helicopter Operating Limits Information System (SHOLIS) developed for the UK Ministry of Defence (MOD) by PMES and Praxis (subcontractor) with Z in the late 1990s and the early 2000s [11].
- The Certification Authority (CA) for Multos smart card scheme developed by Praxis with Z and CSP (Communicating Sequential Processes) [12] in the early 2000s [13].
- The VAL shuttle system for Roissy Charles de Gaulle airport, a fully automatic driverless shuttle servicing the various terminals of Roissy Airport, developed by ClearSy using B method [14]. The VAL system has been operating since September 2006 [15, 16].

The main reason for choosing these case studies is that they cover a wide range of application domains (such as customer transition processing, real-time air traffic control, trusted gateway, ship helicopter operating, smart card certification authority, and airport shuttle transportation) with major formal specification languages such as VDM, Z, B, and CSP. These applications range from the beginning of 1990s to the most recent time (say 2006) which somehow reflects the progress and development of formal applications. Moreover, some of them are applied with the same core specification languages but with different development approaches which makes the horizontal comparisons between these development approaches possible. For instance, the CA and SHOLIS are both developed with Z, the difference between them is that the former does not apply any proof but only rely on traditional testing techniques, while the latter applies proof on system specification and code. In addition, an interesting comparison between formal and informal methods with respect to the same target system is also disclosed by the TG project.

III. Observations

A. Specification Languages

1) Formal specifications cannot stand alone

Formal specification cannot live without a precise, informal definition of how to interpret them in the domain considered [17, 18]. Moreover, formal specification is never formal in the first place, and the properties of a system must first be formulated in a precise natural or semi-formal language such that all parties can speak and understand. For instance, the informal requirements descriptions are the start point of all the case studies mentioned in Section II.

2) Formal specifications are not all purpose

Formal specification and design are effective under some but not necessarily all circumstances [7].

[1] VVSL is a mathematically well defined VDM like specification language with features for modular structuring and specifying operations which interface through a partially shared state.

Developers should restrict the formal specification to certain types of requirements such as the "critical" parts of the system requirements, and, for example, exclude the user interface requirements [1]. For instance, the user interfaces of CDIS and CA are developed in a conventional informal way as shown in Table I.

3) Language Integration may be needed

It is unlikely to just use one formal specification language to specify all possible systems and problems. This is because each specification language has its own built-in semantic bias, and thus it would be impractical to put a language into a domain in which it is not (mostly) suitable for. In practice, developers might need to choose more than one notation and then find a way to integrate and analyze the specifications expressed in different notations. For instance, in the CDIS project, VDM is used for the core specifications, while FSM and CCS are used to specify concurrency and LAN, respectively. And in the CA project, CSP is used for the process design in addition to the core specifications (architecture design and detailed design) specified with Z (see TABLE I).

Note that recently many extensions and variants have been developed for different specification languages. The main intention is to extend the respective languages and to cover a wider range of specification paradigms. Examples of them are VDM++ (object-orientated extension of VDM), Object-Z (an object-oriented extension to Z) [19], Timed-CSP (an extension of CSP which incorporates timing information for reasoning about real-time systems) [20], TCOZ (Timed Communicating Object Z) [21], etc.

TABLE I. SPECIFICATION LANGUAGES USED IN CDIS AND CA

Project	Category	Language
CDIS	Core Specification	VDM
	Concurrency	FSM and VVSL
	LAN	a mix of CCS and VDM
	User Interface	Informal pseudocode
CA	Core Specification	Z
	Process Design	CSP
	User Interface	a user interface prototype was developed and validated with the CA operational staff

B. Code Quality

1) FM may achieve better code in general

In this section, we analyze the code quality from three aspects, namely the numbers of faults (errors that the developer sees) found in the pre-delivered code and the number of changes needed to correct these faults, the number of failures (errors that the user sees) found in the post-delivered code, and the size and speed of the delivered code. We mainly use the statistic data from CDIS [7] and TG [9] to illustrate these issues. The reason for choosing these two projects is that both of them consist of a part of informal development in the projects, such that it is possible to compare the quality between formally and informally developed codes. The difference between CDIS and TG is that the informal code of the former is only developed for user interface subsystem (see TABLE I), while the latter consists of two parallel informal and formal development paths with respect to the same target system.

a) Changes and faults in pre-delivered code

The faults found during unit testing of CDIS disclose that the number of faults normalized by the number of modules in the total formal code is less than that in the informal code (0.46 vs. 0.70) [7]. This could be regarded as one of the positive arguments of formal specifications. However, an interesting thing is that according to Figure 6 (which is borrowed from [7]), the code changes per KLOC (thousand lines of code) of the total formal code in each development quarter are no less than that of the informal code. In other words, it seems that there are more changes in the total formal code during unit testing but with less number of faults. Note that the spikes at or after quarter 4 could be assumed as the onset of system testing.

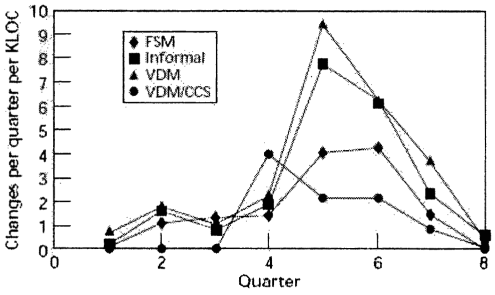

Figure 3. Changes of CDIS reported by quarter, normalized by size of code (source: [7])

It was said in [7] that a fault can result in more than one change to several modules, and thus the total code changes are generally greater than the total faults reported. However, this does not answer the question *why there are less faults but more changes in the total formal code*. One of the possible reasons could be that the formal code is generally more complex than the informal code since the latter only deals with relatively simple part, i.e., user interface. We want to further investigate the cause of the inconsistency, but we could not found additional data about testing and project activity to enlighten us.

b) Failures of post-delivered code

As shown in TABLE II, the CDIS and CA code exhibits remarkably low 0.81 (and 0.75 for the formally developed code) and 0.04 failures per KLOC, compared with the CMM data from [22] and informal implementations from [23]. However, this comparison should be viewed very cautiously, since we do not know the details of the CMM data and the informal implementations.

TABLE II. Average Failure Rates of Delivered Software

Software		Defects per KLOC
CMM Data [22]	Level 1	7.50
	Level 2	6.24
	Level 3	4.73
	Level 4	2.29
	Level 5	1.05
Informal Implementations [23]	IBM normal development	30.0
	IBM Cleanroom development	3.40
	Siemens operating system	6-15
	NAG scientific libraries	3.00
Formal Implementations	CDIS air-traffic control support	0.81 (total) 0.75 (formal)
	CA for Multos smart card scheme	0.04

c) Code size and speed

TABLE III illustrates the code sizes of kernel routine, initialization times, and processing rates of the informal and formal implementations of TG [9]. It indicates that the routine produced by the formal methods is much more succinct, roughly 0.17 times of the informal one. The processing rate for passing a large block of messages of the formally developed code is roughly 13.89 times faster as of the informal one. However, the initialization time needed for the formally developed code is 4.11 times slower as of the informal one. No further explanations are presented in the original literature [9, 10] with respect to this issue. Note that the difference identified here cannot be attributed solely to the use of formal specification, other factors such as experience and ability of the software engineers should also be considered. In the software design stage, engineers with more experience are assigned to the formal group [10].

TABLE III. Code Size and Speed of TG

Measures	Informal Methods	Formal Methods	Rate (F/I)
Size of kernel routine (lines)	371	63	0.17
Initialization time (seconds)	17	70	4.11
Processing rate (chars per sec)	18	250	13.89

C. Cost Analysis

1) Training Cost

a) The training cost of FM is unclear

One of the key worries for applying formal specifications is the training needs for general engineers to read, write, and use formal specifications. Unfortunately, only a few of the case studies have somewhat explicitly mentioned about this issue.

In the TG project [9], it is reported that a basic one week course in the use of formal specification and the IFAD VDM-SL Toolbox is necessary, and a short one or two day supplementary course is needed to meet the specific needs of software designers and implementers. However, adding training in proof would introduce a much greater overhead.

In the CA project [13], it is simply mentioned that the training cost is around 3% of the total effort, and there is no further detailed information about this issue.

In both of the above two case studies, we could not find any data about the experience and background of the engineers. These omitted factors are also important when considering and evaluating the training needs in general.

2) Cost Shift

a) Formal Specifications cause cost shift

Using of formal specification usually adds more cost to the early stage of software development where system requirements are being analyzed and understood. That additional effort, however, can be generally recovered in the later stages of development such as detailed design, coding, and testing.

The cost distribution of the trusted gateway (TG) project [10] confirms this conventional wisdom. In Figure 4, the cost (time) spent for informal and formal methods are normalized by a predetermined total time. In the system specification and architecture design stage, the formal method costs roughly 17% more effort than the informal one ((0.35-0.3)/0.3), while in the detailed design and implementation (coding and testing) stages, the formal method saves roughly 17% and 24% more effort, respectively.

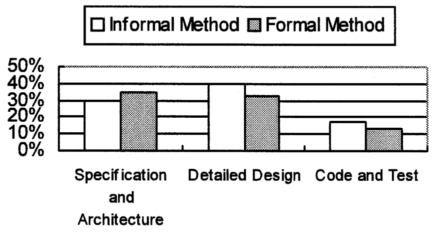

Figure 4. Cost Distribution of TG

The TG project is a good example to illustrate the difference between informal and formal methods, because it is a *comparative* study on the *same* target system with both informal and formal methods. Totally speaking, the formal method saves 7% more effort than the informal one. However, we could not use it directly as one of the evidences of formal methods, since there are many other unclear factors which may affect the result, such as the engineers in the formal one typically have more experience than those in the informal one [10].

b) Proof causes cost shift

One of the key features of formal specifications is that it is possible to carry out formal proof on the specifications (probably with corresponding animation tools in case the specifications are not executable) so as to detect potential errors in an early time of software development. The burden of software testing of conventional development thus can be somewhat relieved. In an extreme case such as the B method, test can be somehow completely replaced by proof, provided that each step of development from original specification to final code has been formally proved.

To illustrate this point, a comparison on the cost distribution between the CA (Certification Authority) project with Z [13] and the VAL shuttle system project with B [15] is presented in Figure 5 and Figure 6, respectively. The CA project relies on traditional testing technique and there is no proof has been carried out on the Z specifications. In contrast, the VAL project follows the B method in which stepwise proof of each specification refinement is required as mandatory. The unit and integration test is removed from the VAL project, and the removing is proposed by RATP, the Parisian Subway Authority, to the manufacturer. However, some global functional tests have been carried out through an independent validation team because the initial requirements description is informal.

The development process of B software can be divided into two phases called Abstract Model (AM) and Concrete Model (CM). The first phase is to formalize every functional requirement from the informal software requirements prescriptions into the Abstract Model containing abstract data types which cannot be directly implemented, and the second is then to build the Concrete Model starting from the non-implementable parts of the Abstract Model. This task is completely systematic and does not require any knowledge on the informal requirements prescriptions [15]. Finally, the last Concrete Model refinement is automatically translated into executable code with some commercial translators. For comparison, we split both AM and CM into two parts, namely AM specification (spec) and proof, and CM specification and proof, respectively. An analog then can be made between the AM spec and proof, CM spec, and CM proof of VAL and the specification and design (S&D), code, and test of CA, respectively.

As shown in Figure 5 and Figure 6, the AM spec and proof of VAL requires roughly 120% more effort than the S&D of CA, while the CM spec and proof of VAL save roughly 7% and 73% more effort than the code and test of CA, respectively. Note that the cost of S&D, code and test of CA and the cost of AM and CM of VAL have the surprisingly same proportion, 79%, in their respective overall development costs.

It should be noted that unlike the cost data of the informal and formal implementations of the same TG target system shown in Figure 4, the cost data of VAL and CA shown in Figure 5 and Figure 6 has not been normalized by a common total time since they are two completely different projects.

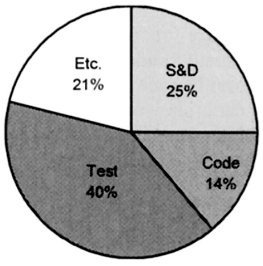

Figure 5. Cost Distribution of CA

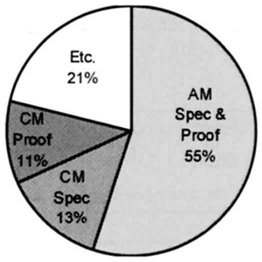

Figure 6. Cost Distribution of VAL

3) Cost-Effectivenss

In this section, we analyze three issues related to the cost-effectiveness of formal methods, i.e., the efficiency for finding faults of proof vs. test, the productivity of FM, and the overall payoff of FM.

a) The evidence for the efficiency of proof than testing is not sufficient

Generally speaking, the cost of proof is much higher than that of specification. This is because the former typically requires more expert intervention and coding skill on the specifier's side. One impression is that commercial pressures work against application of proof where it is not mandated, especially given the shortage of relevant skills among systems engineers [24].

Among the six case studies presented before, three of them, i.e., CICS (with Z), CA (with Z and CSP), and TG(with VDM), do no apply proof on the specifications. In these cases, test seems to be considered as a more cost-effective way than that of proof.

An interesting issue is to compare the efficiencies of fault-finding in different development stages between applications with and without proof. Since we could not find a comparative study on the same one target system which is developed with and without proof in a parallel way, we take the statistic data from the CA [13] and SHOLIS [11] projects to analyze it. The CA and SHOLIS are both developed with Z, the difference is that the former does not apply proof and the latter applies proof on the system specification and code.

The percentage of faults found, percentage of effort (time) spent, and the efficiency (i.e., faults found per effort) in each development process of the SHOLIS project is shown in Figure 7. It indicates that the Z proof on system specifications is the most cost-effective phase for faults finding, which represents that each percentage of effort can find roughly 6.4 percentages of faults. Note that however, this is a kind of "unfair" comparison since the main goals of some processes, such as specification and detailed design & code, are not finding faults but specifying and/or designing the system in a formal way. Nevertheless, the efficiency of proof (e.g., Z proof and code proof) in this project is higher than the efficiency of test (e.g., unit test, integration test, and acceptance test) in general.

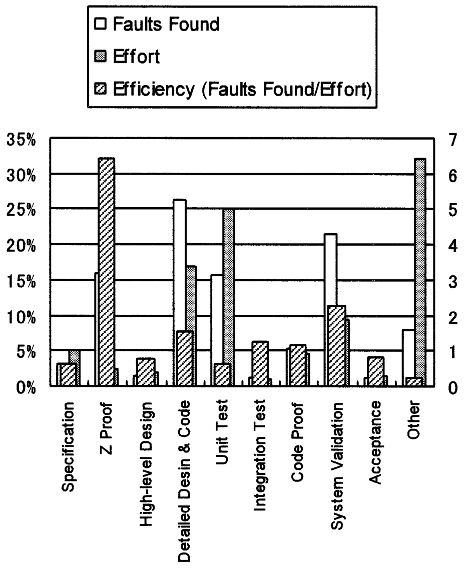

Figure 7. Faults found, effort, and efficiency of SHOLIS

In contrast, since the CA project does not apply proof, the most cost-effective phase for fault-finding is the detailed design and code phase which represents roughly 3.53 times of efficiency (see Figure 8). Although we do not have the data of projects developed with traditional informal methods in hand for comparison, we assume that the test could be the most cost-effective phase as for the informal methods. If this assumption holds, then Figure 4 and 3 also indicate the early fault-finding functionality of formal specifications.

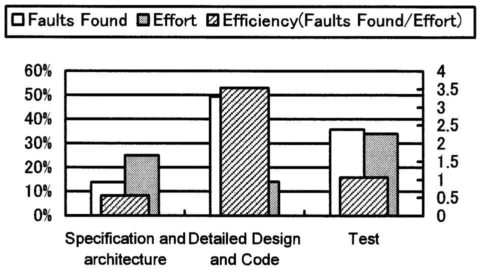

Figure 8. Faults found, effort, and efficiency of CA

By comparing SHOLIS and CA, we get Figure 9, in which we assume that the "sum" of the processes of specification, Z proof, and high-level design of SHOLIS "equals" to the specification and architecture process of CA, and the sum of the processes of unit test, integration test, code proof, system validation, and acceptance of SHOLIS equals to the test process of CA. One reason for making such a synthesis comparison is that we do not have the detailed data of CA about each small development process as SHOLIS.

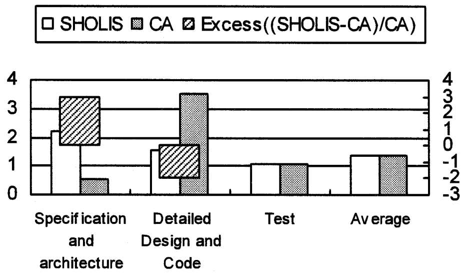

Figure 9. Efficiency comparison between SHOLIS and CA

Figure 9 indicates that in the specification and architecture phase, the efficiency for fault-finding of SHOLIS is roughly 2.96 more times than that of CA. But in the detailed design and code phase, the efficiency of SHOLIS is then roughly 1.99 times less than that of CA. There is no big difference in the test phase in which only 0.02 times of increasing is denoted by SHOLIS compared with CA. The difference of the average efficiencies (of the three phases) between the two projects is trivial, only roughly 0.01 times of decreasing of SHOLIS is represented. Note that the comparison is not based on the absolute values of faults found and effort spent since a part of them is not available in the original literatures [11, 13]. The observations made here thus are quite subjective regardless of other factors of influences, such as they are totally two different projects whose difficulties and efficiencies may not be able to be compared directly.

Several other factors should be taken into account in terms of the comparison of cost-effectiveness between proof and testing, such as the language nature (i.e., whether the specification language emphasizes specification or proof), tool support for discharging proof obligations, the complexity of system architecture, and the limits of formality. Unfortunately, so far there is no sufficient quantitative analysis with respect to the above issues. Further sufficient quantitative analysis on the comparison between proof and test of different applications and specification languages is desired.

b) The productivity of FM is not "clear"

It is reported that the productivities of CDIS [6] and CA [13] are 13 and 28 LOC/Day, respectively. Compared with the industrial standard 7 LOC/Day [4], they represent remarkably 85% and 300% improvements, respectively. In addition, in the CICS project, it is also reported that the productivity of formal methods achieves 9% improvement attributed to less rework during development [4, 3]. However, these data should be viewed very cautiously, since the productivity is an issue highly related to the experience of engineers and the complexity of the target system. We do not find sufficient data and analysis with respect to the above correlated factors, and thus we cannot draw a relatively objective conclusion.

c) The overall payoff of FM is still unclear

As we introduced before, the defect rates of formally developed codes are typically less than those of informal ones, and the productivity of formal methods also seem to be somewhat higher than the average level of traditional informal methods. However, this does not prove that the formal developments are generally superior to the informal ones in terms of the total cost-effectiveness. Special care should be taken into account in terms of the following factors:

- There are so many interwoven factors involved in the case studies, such that it is impossible to allocate pay off from formal methods versus other factors, such as quality of people or effects of other methodologies. Even where data was collected, it was difficult to interpret the results across the background of the organization and the various factors surrounding the application [25].
- The public announcements of success seldom have been accompanied by a complete set of data and analysis, so independent assessment is difficult [7]. For instance, the detailed analysis of training needs of the formal methods is generally missing or not considered comprehensively in such announcements.
- Few comprehensive studies on the comparison of parallel formal and informal developments with respect to the same target systems have been carried out. A notable exception is the BASE TG (trusted gateway) project mentioned early in this article. However, even in this case, the engineers in the software design stage of the formal path were of higher level of skills and experience than those of the informal path, and the final result of the difference on the overall effort was not felt to be significant [8, 9, 10].

IV. C ONCLUDING REMARKS

In this paper, we investigated several industrial applications with formal specifications and presented some comparative observations relating to the specification languages, efficiency, quality, and cost analysis of formal and informal methods. Some of these observations reconfirm traditional wisdoms of formal methods, and some of them propose questions for further discussion and investigation. To conclude these observations, we briefly classify them as two groups as follows.

The observations stating something that are "confirmed":

- Formal specifications cannot stand alone, and they must start and work with sufficient informal documents.
- Formal specifications could be effective under some circumstances such as development of safety-critical systems, but they are not necessarily work under other circumstances such as development of user interface.
- Language integration may be needed because each specification language typically has its own built-in semantic bias so as to be useful for particular domains.
- Formal specifications can usually achieve higher code quality due to its formal semantics. For instance, much fewer defects are found in formally developed code in general and the formally developed code is usually more succinct with higher performance.
- Formal specifications usually cause cost shift to the early development processes, likewise formal proof can also save cost in later testing compared with traditional development methods.

The observations stating something that are "unclear":

- The evidence for the *efficiency of formal proof* than testing is not sufficient so far.
- The *productivity* of formal specifications is not clear, although there are some exciting results have been reported in respective formal applications. However, many influencing factors, such as the training cost and background and expertise of engineers, are generally missing in these reports.
- Related to the productivity problem, the *training cost* is also unclear. Very few of the reports of industrial applications have explicitly mentioned about the training cost. Even where training cost has been addressed, important factors such as the experience and background of engineers are generally missing or not considered.
- Last but most important, the overall *payoff* of formal methods is thus still not clear.

Note again that all of our observations should not ever be regarded as objective and conclusive, since the survey is carried out based on a relatively small sample space and somewhat "incomplete" source data with respect to each analyzing aspect of concern. To understand the advantages and limitations of applying formal specifications in industry in a more comprehensive sense, we are currently carrying out a trial of formal specifications on a small real-world example which has been developed with some traditional informal methods before. Some first-hand material is expected from the trial, especially the comparison between formal and (previous) informal developments, and the difference between different formal specifications with respect to the same target system. In the meantime, another future work is to further investigate a wider range of industrial applications for more facts and deeper analysis based on these facts.

V. REFERENCES

[1] G. K. Palshikar, "Applying formal specifications to real-world software development", IEEE Software, pp. 89-97, Nov/Dec, 2001.

[2] J-R Abrial, "The specification language Z: Syntax and semantics", Tech. report, Programming Research Group, Oxford Univ., 1980.

[3] J. Houston and S. King, "CICS project report: Experiences and results from the use of Z", Proc. of VDM'91 (Berlin), vol. 551, Springer Verlag, 1991, pp. 588–596.

[4] T. McGibbon, "An analysis of two formal methods: VDM and Z", Tech. report, Data & Analysis Center for Software, Aug 1997.

[5] C. B. Jones, Systematic software using VDM, 2nd ed., Prentice Hall, 1990.

[6] A. Hall, "Using formal methods to develop an ATC information system", IEEE Software, pp. 66–76, Mar 1996

[7] S. L. Pfleeger and L. Hatton, "Investigating the influence of formal methods", Computing Practice, pp. 33-43, Feb 1997.

[8] J. S. Fitzgerald, T. M. Brookes, M. Green, and P. G. Larsen, "Formal and informal specifications of a secure system component: first results in a comparative study", FME'94: Industrial Benefit of Formal Methods, M. Naftalin, T. Denvir, and M. Bertran, Eds. LNCS, vol. 873, Springer-Verlag, 1994, pp. 35–44.

[9] P. G. Larsen, "Lessons learned from applying formal specification in industry", IEEE Software Specifal Issue about "Leassons Learned", May 1996.

[10] T. M. Brookes, J. S. Fitzgerald, and P. G. Larsen, "Formal and informal specifications of a secure system component: Final results in a comparative study", FME'96: Industrial Benefit and Advances in Formal Methods, M-C. Gaudel and J. Woodcock, Eds. Springer, Mar 1996, pp. 214–227.

[11] S. King, J. Hammond, R. Chapman, and A. Pryor, "Is proof more cost-effective than testing?", IEEE Transactions on Software Engineering, vol. 26, no. 8, pp. 675–686, 2000.

[12] C. A. R. Hoare, Communicating sequential processes, Prentice-Hall series in Computer Science, Prentice-Hall International, 1985.

[13] A. Hall and R. Chapman, "Correctness by construction: developing a commercial secure system", IEEE Software, pp. 18–25, Jan/Feb 2002.

[14] J-R Abrial, The B book: Assigning programs to meanings, Cambridge University Press, 1996.

[15] F. Badeau, "Using B as a high level programming language in an industrial project: Roissy VAL", Proc. of ZB'05, 2005.

[16] J-R Abrial, "Formal methods in industry: Achievements, problems, future", Proc. of ICSE'06, Shanghai, ACM, May 2006, pp. 761–767.

[17] A. van Lamsweerde, "Formal specification: a roadmap", The Future of Software Engineering (in conjunction with the 22nd International Conference on Software Engineering), A. Finkelstein, Ed. ACM Press, 2000.

[18] A. Hall, "Realising the benefits of formal methods", Formal Methods and Software Engineering, LNCS, vol. 3785, Springer, 2005, pp. 1–4.

[19] G. Smith, The Object-Z Specification Language, Kluwer Academic Publishers, 2000.

[20] S. Schneider, "An operational semantics for timed CSP", Information and Computation, vol. 116, pp. 193–213, 1995.

[21] S. Qin, J. S. Dong, and W-N Chin, "A semantic foundation for TCOZ in unifying theories of programming", Proc. of FME'03, LNCS, vol. 2805, Springer, 2003, pp. 321–340.

[22] C. Jones, Software Assessments, Benchmarks, and Best Practices, Reading, MA: Addison-Wesley, 2000

[23] L. Hatton, "Programming Languages and Safety-Related Systems," Proc. Safety-Critical Systems Symposium, Springer-Verlag, New York, 1995, pp. 48-64.

[24] J. S. Fitzgerald and P. G. Larsen, "Formal specification techniques in the commercial development process", ICSE-17 workshop on Formal Methods Application in Software Engineering Practice (Seattle, USA), April 1995.

[25] S. Gerhart, D. Craigen, and T. Ralston, "Observation on industrial practice using formal methods", Proc. of the 15th International Conference on Software Engineering, IEEE CS Press, 1993, pp. 24–33.