
Project DEPLOY
Grant Agreement 214158
*"Industrial deployment of advanced system engineering methods for high productivity and dependability"*


***DEPLOY DELIVERABLE***

# D47 (D11.5) DEPLOY HOWTO Guide for Managers

***Report – Public***

26 April 2012 – V2.0

FINAL

http://www.deploy-project.eu

# Authors:

Jean-Christophe Deprez (CETIC)
Christophe Ponsard (CETIC)

# Reviewer:

Huang Le Dang (Siemens)
Rainer Gmehlich (Bosch)

# History

| Date | Version | Author | Description |
|---|---|---|---|
| 27/1/2012 | 0.1 | C. Ponsard | Initial structure from writing plan. |
| 12/3/2012 | 0.2 | C. Ponsard | Import material from book chapter |
| 29/3/2012 | 0.3 | J-C. Deprez | Write the initial complete draft |
| 2/4/2012 | 0.4 | C. Ponsard | CETIC review and improvements |
| 2/4/2012 | 1.0 | C. Ponsard | Initial version uploaded to BSCW |
| 4/4/2012 | 1.1_r1 | H. Le Dang | DEPLOY internal review |
| 11/4/2012 | 1.1_r1 | R. Gmehlich | DEPLOY internal review |
| 19/4/2012 | 2.0 | J-C Deprez | Address comments from reviewers |

# Table of contents

# Executive summary

The DEPLOY project worked on improving the maturity of formal engineering methods and tools to raise industry interest and increase take-up. Near Real-World Industry transfers took place, first with the DEPLOY industrial partners, then joined by DEPLOY associate partners. In DEPLOY, Industry transfer initiatives were referred to as *Deployments*. Other successful applications of formal methods also took place among the DEPLOY industry group and service-oriented partners.

DEPLOY devoted a significant amount of effort to collect information on these various Industry deployments and to present them as evidence of formal methods transfer to Industry. It is expected that these pieces of evidence can also be exploited by companies not involved in DEPLOY and to help them make informed decisions regarding formal methods adoption. DEPLOY confirmed the key role played by managers in this adoption process, either at project, Q&A or strategic level.

This document provides a guide for managers who consider experimenting and applying formal engineering methods. It is based on the lessons learned from deployments of formal methods to the DEPLOY Industry partners. It covers a range of questions deemed of interest to managers found in the Evidence repository at www.fm4industry.org. Although this work was primary driven by DEPLOY specific methods and tools (i.e. Event-B and Rodin), these guidelines also reference a number of other methods and tools and, more importantly, they are expressed in a more general context common to all of them.

Managers reading this guide are also invited to consider accessing the website above to enjoy an enriched experience with cross-links among related questions as well as success stories. They will also find updated material as the repository is regularly being updated to stay current with the latest evolution such as the release of new standards, e.g., DO-178C, ISO 26262 or to report on more recent industrial applications of formal methods provider by any enterprises after the end of DEPLOY. Therefore, managers are invited to collaborate on the content of this website, not only to comment on the existing material but also to provide information related to their experimentation with formal methods.

# 1 Introduction

This guide addresses managers' foremost concerns regarding formal methods. Since their inception, formal methods have had the reputation of being a topic that does not scale to Industry's needs and only applicable by theoreticians on toy example. Furthermore, it is often believed that formal methods cannot be learned easily, completely disrupt product development lifecycle and cannot be applied together with traditional software development approaches.

This guide hopes to provide evidence regarding these formal method myths [1] so that managers become knowledgeable enough to better grasp the pros and cons of formal methods.

This guide is an excerpt from the body of evidence presented at:

http://www.fm4industry.org

Questions of most interest to managers where significant evidence can be made in the answer are presented in this document. However, we encourage managers to access the site above to access the full list of questions and the latest updates.

The remaining of this document is organised as follows. After a brief overview of managers' influence on the adoption of formal methods in Section 2, a list of questions and answers on important aspects related to formal method usage in Industry are presented in Section 3. Industry context is too divers to provide a step by step guide on transferring formal methods in an organisation. Instead, with its practical question/answer format this guide provides an easy, ad-hoc reading approach for managers. Each question answer is self-contained including its external references. A reader may therefore jump randomly back and forth through Section 3. Finally section 4 gives concluding remarks.

# 2 The manager role in the adoption of formal methods

It is a manager's world. Thus decisions regarding development lifecycle paradigm are in the hand of managers.

Managers at different levels, in different sectors and managing different disciplines of product and system development may be interested to learn about formal methods. Furthermore, managers will also want to understand how formal methods could affect their team of engineers, analysts, developers and testers. To meet this broad scope of interest, this guide attempts to address questions on a variety of viewpoints.

From interactions with managers from the DEPLOY Industry partners as well as external to the consortium, it was notice that managers often feel that they are in unique situation. They then infer that this uniqueness will make formal methods incompatible with their organization settings.

To help to jump this first hurdle, we refer to the survey of Woodcock et al. that presents

information related to the application of formal methods on 62 significant Industry cases [2]. This survey highlights three important points. Formal methods apply to:

- Many different sectors such as transportation, defence, financial, telecom, nuclear, healthcare, office and administration, consumer electronics, etc.
- Many different types of application such as real-time, distributed, transaction processing, high data volume, control engineering, parallel, hardware, CASE tools, service oriented architecture, etc.
- Many different application sizes from 1K lines of code to systems as large as 1M lines of code.

Although these survey data should convince almost everyone that others have used formal methods in their sectors, for their type of applications and for similar application sizes, in many cases, managers may still feel that these general types of surveys do not apply to their specific organizational context.

In the remaining of this section, we therefore go one step deeper and present the diversity of organizational contexts of each Industry partner involved in the DEPLOY project, namely, Bosch (Automotive), SAP (Business), Siemens (Mass Transport), Space System Finland (Space). Furthermore, we also explain the different visions that each of these organizations has for the application of formal method to fit in their respective context.

## 2.1 Organizational contexts and formal method deployment strategies at DEPLOY Industry Partners

The contexts and strategies for formal method deployment at DEPLOY Industry partners will be depicted on the following aspects:

- The organization unit of the Industry partners involved in DEPLOY
- Their general vision on how formal methods should be used in their company (at what stages to apply formal methods , whom and how many staff members to train)

The actual units of the four Industry partners are composed of two R&D units of large organisations, namely, Bosch and SAP, one safety analysis unit at Siemens and one whole SME, Space System Finland (SSF). Thus, in terms of organisation size, three units of large companies and one small enterprise are involved. It is worth highlighting that where Bosch and SAP involve their R&D units, Siemens involves safety engineers that are part of the production department.

Independent of company and unit sizes, all four Industry partners directly staffed a similar number of employees on the DEPLOY project, that is, from 2 to 3 people. Most Industry people working on DEPLOY had a previous experience with formal methods but only few were well versed on proof-based methods such as event-B, which was the main formalism promoted during the DEPLOY project.

Except for two participants from SSF and one at Siemens, all others directly involved in

DEPLOY held a doctorate degree. While the Siemens engineer had experience in the B formal method, the two participants from SSF without a doctorate degree did not have an experience with any formal methods prior to DEPLOY.

With regards to strategies for the deployment of formal methods in their organisation, each of the four companies had a different view.

SSF has built a part of its business activities on formal analysis most notably in the space sector applied on various European Space Agency projects. While analysts and architects use formal methods, developers needn't handle any formal methods. The involvement of SSF in DEPLOY was primarily to learn a new formal method, namely, event-B. A secondary goal was to determine if developers with no prior experience could learn event-B, at least to be able to read and understand formal specifications in the event-B language.

The main rationale for the involvement of Siemens safety engineering team in DEPLOY is to further their expertise in the family of B formalism. Currently, Siemens is already using the B formal method to model their software specifications. Thanks to event-B, safety engineers will raise the level of reasoning at the system level and will be able to mathematically prove system safety properties.

At SAP, the main objective is to use formal methods for generating test cases for testing business scenarios, in other words, to conduct model-based testing. In this context, SAP management expects the application of formal methods to be completely transparent to testers who should keep using their traditional tool for specifying test cases.

In short, SSF and Siemens expects their production engineers to master formal methods where SAP expects that their production testers will not be made aware of the underlying formal method. The management at Bosch takes an in-between stance.

Bosch wants to explore if formal methods will enable to better handle the increasing complexity of automotive computing. This complexity currently requires a massive effort in test-based verification. The Bosch R&D unit wants to determine if formal methods can increase productivity. When transferred to Bosch business units, the application of formal methods should be hidden from engineers and only few experts in the business units should be trained and be capable to handle the work required to apply formal methods.

## 2.2 Conclusions of Deployment Results at Industry Partners'

Section 3 and the whole evidence repository at www.fm4industry.org detail pieces of evidence at a fine-grained level. However a few general statements can be made with regards to the success of DEPLOY in deploying formal methods (mostly connected to the Event-B and ProB formalisms)

Although Industry partners had different contexts and strategies, DEPLOY could successfully address partner's expectations and demands. As such DEPLOY in itself is a significant evidence that formal methods can be successfully used in various Industry domains meeting Industry needs of partners with different strategies for the application of formal methods and widely different contexts.

In general, the following aspects are worth highlighting:

- **Application of formal methods at different stages of the development lifecycle is possible**. The four Industry partners had different visions on who will use formal methods and how. Furthermore, partners did not target the same stages of the development lifecycle. Some mainly focus on the requirement and design stages while other targeted the verification stage. In all case, DEPLOY could satisfy their needs.
- **Formal methods can be applied by various group, department or unit of an organisation.** During DEPLOY two of the four Industry partners involved engineers from their production department while the other two partners involved R&D units. Although mentality, approaches and vision are different in R&D vs production/business units, both could apply formal methods successfully.
- **Application of formal methods neither requires a Ph.D. level education nor previous experience with other formalisms, although familiarity with other formal methods quickens the learning curve but the lack of prior experience is definitely a surmountable hurdle.**
  - **Previous experience not required**: all but two Industry participants have past experience with formal methods. However, even those with experience were unfamiliar with the proof-based approach proposed in the B formalism. Only Siemens had experience with the B formal method. The other participants had experience with model checking and other formalisms where the difficulty lies in modelling but not proving.
  - **No PhD required**: although most DEPLOY Industry organisations involved people with doctorate degrees a few participants did not. DEPLOY showed that software engineer without Ph.D. and without formal method background could be trained fairly quickly to become proficient in reading formal specifications and conceiving simple models.

**This diversity of contexts of the four organizations involved in DEPLOY together with the diversity of uses of formal methods in these organizations should definitely convince other managers that, as unique as their situation may feel, formal methods can be unequivocally applied to their context.**

### References – Sections 1 and 2

[1] Anthony Hall. 1990. Seven Myths of Formal Methods. *IEEE Softw.* 7, 5 (September 1990), 11-19. DOI=10.1109/52.57887 http://dx.doi.org/10.1109/52.57887

[2] Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. 2009. Formal methods: Practice and experience. *ACM Comput. Survey.* Vol 41, nr 4, October 2009

# 3 Levering on a body of evidence on formal methods

During the DEPLOY project, evidence related to formal method transfer to Industry and usage by Industry were collected from various activities undertaken by Industry partners. Rather than supervise all possible actions performed by Industry partners and their interactions with Academics, a set of themes of interest to Industry were identified and evidence material was collected only for these themes. However, the evidence collection went beyond the scope of the DEPLOY project and a fairly extensive literature review collected evidence material from research projects other than DEPLOY.

The themes of interest to Industry identified by DEPLOY Industry partners are the following:

- General topics of concerns related to formal methods in Industry
- Impact on an organisation with regards to *training scope and resourcing*
- External factors advocating take-up of formal methods (competition, standard bodies, laws, etc.)
- Understanding the *impact on* the Software/System Development *Process*
- Known strengths and weaknesses of tools associated to a formal method as well as the quality of support by tool providers

The full list of Questions/answers on these themes are published at

http://www.fm4industry.org

Below we present excerpts of questions and answers with most interest to managers.

*NOTE: To make each question/answer self contained, citations and references are listed directly after the text of the question/answer.*

## *3.1 General topics of concerns in Industry on formal methods*

Although this theme is very broad, a few interesting points can be addressed to answer the following questions:

- Why have formal methods failed to breakthrough on the market for such a long time?
- What important system concepts can be handled "elegantly" with a selected formal method?
- What interesting books have been published on the application of formal method in Industry?
- What other initiatives exist on collecting data regarding formal methods and their application in Industry?

### *3.1.1 Why have formal methods failed to breakthrough on the market?*

Following the rapid growth of computer science, exaggerated claims about formal methods might have been made in the past by proponents of formal methods [1]. Such

claim might have hindered the credibility of proponents, and of formal methods themselves. As an analogy, consider the near- science-fiction atmosphere that was around the rapid growth of artificial intelligence in the '80, and how it has been reconsidered since then.

More precisely, in his seminal paper [2], Anthony Hall identified seven myths about formal methods. Seven additional myths have been identified later by Bowen and Hinchey [3]. These are false believes that industrials have about formal methods, and that considerably hinder Industry adoption [4].

The first seven myths are as follows [2]:

- *Myth 1: Formal methods can guarantee that software is perfect.*

  Actually, formal method can prove or verify if an artefact has a set of predefined properties. If these properties are incomplete or inaccurate with respect to the actual requirements, or if the requirements are themselves incomplete or inaccurate, formal methods will not compensate such weaknesses.

- *Myth 2: Formal methods are all about program proving.*

  It is true that program proving has been a very long-run topic in the field of formal methods because source code is an existing artefact exhibiting the features of formal language, including a notation that can be analysed by formal tools, and a nearly-formal semantics. However, Formal methods can also verify artefacts other than programme code such as system-level models, as done in the DEPLOY project, communication protocols, and requirements models, etc. It is true that in DEPLOY proving remains a significant aspect. However, that are yet formal methods that do not require proofs to be handle by the users. Notably, model checkers are successful at verifying particular properties such as deadlock freedom only by running a tool with input models.

- *Myth 3: Formal methods are only useful for safety-critical systems.*

  Formal methods tend to provide a very high level of assurance about some properties; and this is especially desirable in safety-critical systems. However, other sectors have high assurance requirements, such as business, security-critical systems, etc. In the DEPLOY project, SAP and Space System Finland were deploying formal methods. Furthermore, Formal methods were also used in DEPLOY verifying instruction set of chip architectures at XMOS –XMOS was an associate DEPLOY partner. So, at SAP, SSF and XMOS, the situation was business-critical rather than safety critical. The choice of using formal methods or not is more tightly related to high assurance than to safety.

- *Myth 4: Formal methods require highly trained mathematicians.*

  The mathematics behind formal methods is easy. What is difficult is the engineering. Formal methods require one to precisely specify and reason about the system properties, etc. This level of precision tends to be much higher than the one reached in written specifications, hence more difficult to reach. Training delays are generally estimated to one or four weeks, with a trained teacher. It is true that formal methods are about mathematics, and that majority of people tend to dislike mathematics,

quickly sticking a "too difficult" stamp on it. In some cases, it is also possible to hide formal methods behind domain-specific languages, for example, hiding mathematical notations behind graphical notations à la UML.

- *Myth 5: Formal methods increase the cost of development.*

  Formal methods shift the cost of development from late testing phase to early analysis phases. Managers tend to rely on the feeling that something is being developed to estimate the progress of their teams. Writing source code immediately tend to be less frustrating that developing models during as much as 30% of the project lifetime, depending on the technique in use. It is also true that using formal methods dramatically changes the rate of identification of issues at each phase of development cycle.

- *Myth 6: Formal methods are unacceptable to users.*

  Formal specifications help users understand what they get, simply because they precisely state the properties that are proven on the artefact. Formal specifications can be difficult to communicate in raw manner to non-trained individuals; just like reading source code might be difficult for non-computer scientists. Several techniques can be used to help this understanding, including: paraphrasing the formal specification into natural language (which turns to be quite easy), animating the formal specification (depending on the available animation technology), and demonstrating the consequences of the formal specification. In many cases, users do not care whether a formal method was used or not. They are mostly concerned with the end system and its usability rather than the process used to develop this end system.

- *Myth 7: Formal methods are not used on real, large-scale software.*

  Several success stories from the work performed on Industry size projects by DEPLOY partners are mentioned in the online FAQ at http://www.fm4industry.org. Other reference cited through the online FAQ point to other success stories by organisation external to the DEPLOY project.

The additional seven myths are the following [3]:

- *Myth 8: Formal methods delay the development process.*

  This myth has a relationship with Myth 5. In general, once formal methods are mastered by the engineering team, the development time is roughly similar when applying formal or informal methods. It is however true that using formal methods dramatically changes the effort allocation at the different stages of the lifecycle and that the rate of identification of issues at each phase of development cycle is also altered. In particular, formal modelling often delays the requirement and design stages but it enables the discovery of bugs in these early stages of a project when problems and error are much cheaper to solve.

- *Myth 9: Formal methods are not supported by tools.*

  Formal methods require specific tools to be developed to handle them such as theorem provers, model-checkers, etc. This technology is of paramount importance

for the successful deployment of formal methods, as it provides the degree of automation to make formal methods attractive [5]. Obviously, it is quite frustrating to use a method that requires one to perform many manual systematic checks while one may have the feeling that a machine would perform such checks better and much faster. The technology that is needed to deploy formal methods with the adequate automation is becoming mature, and as the computing power available to common desktop platform is increasing as well, such tools can be developed and provide direct support to engineers who needn't access resources other than their own workstations or laptops.

- *Myth 10: Formal methods mean forsaking traditional engineering design methods.*

  Formal methods provide a formal notation, also known as "formal specification language", and some form of deductive apparatus, also known as "proof system". As such, they are not actual development methods; they should be called tooling or techniques. They will not make up for the engineer skills and know-how, and they need to be integrated in some development process, just as engineer learned to use calculators in the very early days of computer science.

- *Myth 11: Formal methods only apply to software.*

  Formal methods apply also to hardware or system engineering. Formal methods reason on some input that need to be made precise enough to be amenable to automated analysis. Software artefacts tend to match this condition since programs are executable by a machine. However, in DEPLOY, event-B is used to model whole system and proof certain properties of this entire system.

- *Myth 12: Formal methods are not required.*

  Formal methods deliver a very high assurance level on the artefact they analysed about the property that was analysed. This high level of assurance might sometime be considered as overkill, but this is to be considered on a case-by-case basis. It is true that sectors exhibiting some degree of criticality tend to adopt formal methods more easily than sectors without such criticality. In a situation where time to market is more important than reliability and safety, formal methods should not be used. However, if at later time, a product even in a non-safety critical domain breaks through and generates huge revenues for an enterprise, it may then become important to verify certain properties of new version of the product before launching upgrades. SAP is a good instance of this situation.

- *Myth 13: Formal methods are not supported.*

  Tools support for formal methods has gained significant maturity in the last decade. Many model checkers have been released or embedded inside larger tool suites, for example, Coverity Prevent, Polyspace, Prefast in Microsoft Team Foundation Sever. In addition, several companies have flourished around formal method tooling and application, for example, Clearsy and Systerel provide support on tooling and also on modelling on the B-family formalism, ESTEREL provide help with SCADE, Polyspace and Coverity provide training and support for their respective source code analyzers. Finally, given the tighten completion, Europe is slowly encouraging the Academic world to provide help to industry, either through applied research projects

under FP7 or even directly offering consultancy services to Industry.

- *Myth 14: Formal-method people always use formal methods:*

  Formal-method experts recognized that formal methods are not a silver bullet. In particular, formal modelling of an application where time to market is crucial would not be practical. Furthermore not all properties can be formally checked on a given system in a practical way. For instance, the run-time of some source code is estimated through benchmarking, user interfaces are tested manually, etc. Formal method people tend to generally focus on a fragment of the whole system focusing on the most critical, the most complex, or the most amenable to formal analysis.

From our experience in the DEPLOY project, it seems that industrials tend to avoid changing their development process because of the incurred cost: switching people to a new technology takes time, some people do not adapt and are therefore wasted, there is a risk in the first project, etc. The biggest driver in industry is money. The second biggest driver is when they are obliged to do something, e.g. to comply with some regulation. For instance, the famous line14 metro that was developed a decade ago was developed using the B formal method because the customer explicitly required this method to be used. A third reason from our DEPLOY experience is that a manager might be blamed for a failure due to his attempt to deploy new "exotic" development methods such as formal methods, but he will less likely to be blamed for a failure arising because he pushed his team to keep using methods that were successful so far.

**References**

[1] Michael G. Hinchey. 2003. Confessions of a formal methodist. In Proceedings of the seventh Australian workshop conference on Safety critical systems and software 2002 - Volume 15 (SCS '02), Peter Lindsay (Ed.), Vol. 15. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 17-20.

[2] Anthony Hall, Seven Myths of Formal Methods, IEEE Software, pp. 11-19, September/October, 1990.

[3] Jonathan P. Bowen and Michael G. Hinchey. 1995. Seven More Myths of Formal Methods. IEEE Softw. 12, 4 (July 1995), 34-41. DOI=10.1109/52.391826 http://dx.doi.org/10.1109/52.391826

[4] Michael G. Hinchey, Encouraging the Uptake of Formal Methods Training in an Industrial Context, in Leveraging Applications of Formal Methods, Verification and Validation Communications in Computer and Information Science, 2009, Volume 17, Part 9, 473-477, DOI: 10.1007/978-3-540-88479-8_33

[4] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal methods: Practice and experience. ACM Comput. Surv. 41, 4, Article 19 (October 2009), 36 pages. DOI=10.1145/1592434.1592436 http://doi.acm.org/10.1145/1592434.1592436

### 3.1.2 Important system concepts "elegantly" handled with formal methods

The goal of formal methods is to verify, test, or develop a model that has desired properties.

- The properties can be either generic domain-independent such as absence of deadlocks or live-locks or domain-specific, possibly in the overall environment, such as an invariant or a deadline for a reaction of the model.
- The model can represent a finite state machine, a more complex transition system such as a communication protocol, or a hybrid system (with both discrete and continuous behaviours), etc.

The main issue in constituting a design flow is selecting the right formal methods [1]. These can vary greatly according to the kind of properties to prove on the targeted system. Furthermore, it is crucial to ensure that the property proven on the model makes sense in the real setting of the system.

For instance, when a property of a reactive system is verified, it is crucial for the inputs considered by the model to cover all possible inputs delivered or generated in the real world environment in which the product will operate. This can include timing issues, values, etc. For high assurance system, one should also consider failures of various components of the environment and erroneous inputs.

Different formal method natively support different system concepts such as real time, invariants, etc., however, one might twist a given formal method to incorporate additional concepts that it does not support natively. This twisting increases the risk to produce inelegant models, that is: hard to understand, maintain, requiring unnecessarily complex proofs etc.

In the remaining of this section, we cover several aspects of system modelling such as discrete vs. continuous, real time, and probabilities. We present facts to help Industry members to select the most appropriate method for each of them, according to their specific industrial and technical setting.

***Reasoning about the real world: continuous world vs. discrete model***

For instance, the B formalism captures only discrete values. Consequently, the physical dimensions captured in B models to represent the environment of the system under construction are all made discrete. This has to be taken into account when we reason about concepts that are continuous in the real world, such as the speed or distances of metro's in the context of metro driving software. In such context, two main strategies can be developed, and used together:

- Chose a *discretising* that is fine enough, so that the correctness of the concrete implementation can be deduced by the correctness of the discrete model. Speed of trains can be represented in cm/sec instead of using the traditional km/h dimensions.

- Develop a model that uses safe rounding techniques, so that the discrete values are always rounded "on the safe side", even for intricate arithmetic functions [2]. For instance, if we prove that trains will always stop within the available stopping distance rounded down to the inferior meter, the proof makes sense in the real continuous world.

A more intricate example is when the controlled system obeys complex physical laws, as met in flight control systems such as the Traffic alert and Collision Avoidance System (TCAS). In such system, one wants to prove that airplanes will not collide when they are under the control of software that pilots the plane to perform collision avoidance manoeuvres. The trajectories followed by airplanes must always be flyable, that is: airplanes cannot brake; make sharps turns or climb abruptly [3]. Such systems are called hybrid systems, and a specific research with specific tools has been done to handle such cases in a completely sound manner [4][5][6].

***Real time***

A more common issue in verification is real time. Real time is when the reaction of the software must obey timing constraints. The most typical constraints are deadlines, where the output of the software must come before some timeout, but other constraints can also appear such as early deadlines or concomitance where the software cannot produce an output before some units of time or where it must produce two outputs at the same time or within some bounded time interval.

Model-checking is a technique that is able to verify complex timing properties on transition systems. Most of these tools reason on models where the time is made discrete, either because it is structured along a "universal clock" that regularly ticks, and generates a new state of the system or because time progresses by one unit at each event of the considered model.

- SMV, and its successor NuSMV are well-known representative of such formal tools. In their logics, time is discrete, and progresses along a universal clock. They support finite state machine. Properties to verify can be expressed in temporal propositional logics [7]. For instance, one can specify that a train must stop within three "tics" after some signal is received as follows:

  `StopSignal => ooo TrainStopped` where `o` represents the "next" operator that captures a "tic", and => represents an implication.

- UPPAAL is a well-known model checker that supports the notion of continuous time. In the logics of this tool, time is continuous, and progresses as in the real world, while modelled system is a finite state machine whose conditions can be expressed in term of the continuous time [8]. For instance, on can specify the same train stopping requirement with better realism as follows:

  ```
  StopSignal => <>_{3 minutes} TrainStopped
  ```

One does not always need to formally reason on timing conditions even if a formal method is in use. In the DEPLOY project, Siemens has reported that their control system are conceived as a single response loop that is executed repeatedly and outputs are computed in a single pass of this loop [9]. The loop itself must execute within some time

bound. This time bound is checked at run time, and in case the deadline is missed, the control system activates the emergency brakes, which is the default emergency procedure that brings the train into a safe and stable state. This approach can only support a single deadline, and can only be applied if a default safe procedure can be activated at any time.

Bosch has tried to reason about responsiveness of reactive systems modelled in Event-B. Event-B does not natively supports the notion of deadline and timing condition; rather it is aimed for reasoning about invariant, convergence, and refinement of state machines. The approach of Bosch was to adding a set of time counters within their models, and incrementing the counter at each relevant event, to model the elapsing of time. In order to prove convergence and responsiveness, they also had to develop a series of complex invariants referencing the counters from the model to capture the exact semantics of these counters [10]. This approach requires a lot of effort in the development of invariants that capture the notion of time, and delivered very complex model, that is, hard to understand, maintain, etc. Similar patterns for reasoning about time and consistency in business information systems in Event-B have been proposed in [11]. These patterns rely on a specific variable representing a time counter, and specific event called "tic" to model the elapsing of time.

There have been proposals to integrate the Event-B method with the SMV and UPPAAL tools in order to natively support the notion of time in the Event-B language [12].

However, the initial semantics of Event-B does not support the notion of responsiveness; this integration therefore required extending the language and part of its semantics.

***Fault tolerance***

There are several ways to reason about fault tolerance using formal methods. It depends mainly on the kind of property one wants to prove about the verification target, and on the kind of error one wants to envision.

When one wants to reason about the robustness of the verification target against some well-identified failures in the environment, a common technique is to model not only the target of verification, but also its environment. Modelling the environment allows one to capture its possible faults into the model, and to reason on the overall system. This approach has been followed during the DEPLOY project in the context of business information systems [13] [14]. They developed Event-B models that included not only the application that was the target of verification, but also part of the environment, namely the underlying communication middleware.

When one requires quantification of some overall error rate of a system, one needs to include probabilities into the model.

***Probabilities***

Specific probabilities for relatively simple systems can be estimated manually or with the help of common statistic tools [15]. For more complex systems, specific tools can be used to formally reason on them.

PRISM is a good representative of probabilistic model checker [16]. Probabilistic model checking is a formal verification technique for the modelling and analysis of stochastic systems. It has proved to be useful for studying a wide range of quantitative properties of

models taken from many different application domains. This includes, for example, performance and reliability properties of computer and communication systems. It natively supports input models representing state machine with probabilistic transitions.

PEPA is a formal language that allows quantitative analysis of systems. Models can be described in a compositional way - as cooperation between individual automata that carry out actions. By associating rates to each action of the model, the description is interpreted against an interleaving semantics that gives rise to an underlying Continuous Time Markov Chain. Typical questions that can be answered by the analysis of the Markov chain are the throughput of an action or the utilisation of a component [17] [18].

There have been attempts to reason on overall probability of failure of systems in Event-B. Event-B does not natively support the notion of probability.

**References**

[1] Arvind, Dave, N., and Katelman, M. 2008. Getting Formal Verification into Design Flow. In Proceedings of the 15th international Symposium on Formal Methods (Turku, Finland, May 26 - 30, 2008). J. Cuellar, T. Maibaum, and K. Sere, Eds. Lecture Notes In Computer Science. Springer-Verlag, Berlin, Heidelberg, 12-32. DOI= http://dx.doi.org/10.1007/978-3-540-68237-0_2

[2] Butler, R. W., Carreño, V., Dowek, G., and Muñoz, C. 2001. Formal Verification of Conflict Detection Algorithms. In Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (September 04 - 07, 2001). T. Margaria and T. F. Melham, Eds. Lecture Notes In Computer Science. Springer-Verlag, London, 403-417.

[3] André Platzer and Edmund M. Clarke., Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, 16th International Symposium on Formal Methods, FM, Eindhoven, Netherlands, Proceedings, volume 5850 of LNCS, pages 547-562. Springer, 2009. (c) Springer Verlag

[4] André Platzer. Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, 2010

[5] Alur, R., Henzinger, T.A., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Trans. Software Eng. 22(3) (1996)

[6] Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In Morari, M., Thiele, L., eds.: HSCC. Volume 3414 of LNCS., Springer (2005) 258-273

[7] NuSMV: a new symbolic model checker http://nusmv.fbk.eu

[8] Kim G. Larsen and Paul Pettersson and Wang Yi, UPPAAL in a nutshell, Int. Journal on Software Tools for Technology Transfer, vol 1, 134-152, 1997

[9] R. De Landtsheer, C. Ponsard, Evidence Report for STS – May 2009, Internal Document, http://www.deploy-project.eu, July 2009

[10] Felix Lösch, WP1: Event-B Modelling Strategy, DEPLOY Plenary, October 2010

[11] Bryans, Jeremy W. and Fitzgerald, John S. and Romanovsky, Alexander and Roth, A. (2010) Patterns for Modelling Time and Consistency in Business Information

Systems. In: 15th IEEE International Conference on Engineering of Complex Computer Systems. Oxford, UK. March, 2010. IEEE Computer Society.

[12] Alexei Iliasov, Linas Laibinis, Alexander Romanovsky, Elena Troubitsyna, Towards Real-time Verification Support for Event-B, DEPLOY Plenary, October 2010

[13] Iliasov, Alexei and Romanovsky, Alexander (2008) Refinement Patterns for Fault Tolerant Systems. In: EDCC 7: the Seventh European Dependable Computing Conference (EDCC-7), May 7-9, 2008, Kaunas, Lithuania.

[14] Bryans, Jeremy W. and Fitzgerald, John S. and Romanovsky, Alexander and Roth, Andreas (2009) Formal Modelling and Analysis of Business Information Applications with Fault Tolerant Middleware. Proceedings 14th IEEE International Conference on Engineering of Complex Computer Systems ICECCS 2009. . pp. 68-77

[15] Michele, Mazzucco and Manuel , Mazzara and Nicola, Dragoni Design of QoS-aware Provisioning Systems. In: 4th Nordic Workshop on Dependability and Security (NODES 2010), Copenhagen, Denmark.

[16] M. Kwiatkowska and G. Norman and D. Parker, PRISM: Probabilistic Model Checking for Performance and Reliability Analysis, ACM SIGMETRICS Performance Evaluation Review, vol 36, no 4, 40-45, 2009

[17] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, number 794 in Lecture Notes in Computer Science, pages 353-368, Vienna, May 1994. Springer-Verlag.

[18] J. Hillston. Process algebras for quantitative analysis. In Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05), pages 239-248, Chicago, June 2005. IEEE Computer Society Press.

### *3.1.3 Books on formal method in Industry*

Showing a regain of interest in formal method by Industry, two books will be published in 2012 on formal method in Industry practice. The first one is mainly focus on the B and event-B formalism, a proof-based formal modeling approach while the second book focuses on model checking.

- *Industrial deployment of system engineering methods providing high dependability and productivity.* A. Romanovsky, M. Thomas (Eds). Springer. 2012
- *Formal Methods for Industrial Critical Systems: A Survey of Applications.* Stefania Gnesi, Tiziana Margaria (Eds), Wiley, March 2012, Wiley-IEEE Computer Society Press, 304 pages, ISBN: 978-0-470-87618-3

A few other interesting books published in a more or less recent past:

- *Software for dependable systems: sufficient evidence?* Daniel Jackson, Martyn Thomas, and Lynette I. Millett, Editors, Committee on Certifiably Dependable Software Systems, National Research Council, 148 pages, ISBN: 0-309-66738-0.
- *Industrial-Strength Formal Methods in Practice,* Hinchey, Springer London Ltd,

1999

Surveys on formal methods in Industry, although the second one is not a book its does reports on 62 Industry use cases.

- *Industrial applications of formal methods to model, design, and analyze computer systems: an international survey,* Dan Craigen, Susan Gerhart and Theodore L. Ralston, Ed. Noyes Data Corp, 1993.
- Woodcock, Jim and Larsen, Peter Gorm and Bicarregui, Juan and Fitzgerald, John S. (2009) Formal Methods: Practice and Experience. ACM Computing Surveys, 41 (4). pp. 1-36.

For a general introduction to formal methods

- *Rigorous Software Development: An Introduction to Program Verification,* José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, Simão Melo de Sousa, Springer, 29 déc. 2010 - 307 pages.
- Understanding Formal Methods, Jean François Monin and Michael Gerard Hinchey, Springer London Ltd, 2003, 275 pages.

Besides generic books on formal methods, additional book targeting a single formalism have been published. This document does not intent to promote one formal method over others. Thus books on a particular formalism are intentionally not listed in this answer.

### 3.1.4 Other initiatives on collecting data regarding formal methods

Other interesting initiatives related to formal methods and their application in industry are the following

- Formal Method Europe (FME, URL: http://www.fmeurope.org). FME aims to encourage formal methods research and application. Activities include the dissemination of research findings and industrial experience though our symposia and sponsored events; the development of information resources for educators.
- Open-DO (URL: http://www.open-do.org). Open DO is an Open Source initiative that aims to create a cooperative and open framework for the development of certifiable software.
- Formal Methods Wiki (URL: http://formalmethods.wikia.com). It provides an extensive list of pointer to project, repositories and people connected to formal methods. In particular, companies using formal methods are listed at http://formalmethods.wikia.com/wiki/Companies
- ERCIM Working Group on Formal Methods for Industrial Critical Systems (FMICS, URL: http://www.inrialpes.fr/vasy/fmics/). This site contains many links to European organisations involved in formal methods research as well as ERCIM newsletters published on topics related to formal methods.

## 3.2 Impact on an organisation with regards to training scope and resourcing

Instead of answering particular questions on train, we merely relate how the DEPLOY project performed its transfer activities. Although there may be other training approaches, the one followed by the DEPLOY project satisfied the Industry partners and it provides a convincing case for other to rely on.

At a high level, the DEPLOY project followed a similar approach to transfer the event-B formal method to all Industry partners. The steps below were followed:

1. A short intensive training session (three to five days). Out of this training session trainees are able to specify simple system in Event-B and to read fairly complex models specified by experts.
2. Identification of pilot projects for each partner. The pilot projects should represent a small real-world problem on which the company previously worked not using formal methods. This pilot project was then broken down in an even smaller mini-pilot project targeting a small part of the pilot project.
3. Industry partners attempt to solve the mini-pilot requesting coaching from academic experts on need-to basis.
4. Once a sufficient expertise on formal modelling and proving is acquired, the partner then worked on the complete pilot project in an autonomous way only asking for advice from academic experts if a complete blocking point was reached and could not be solve even after a significant effort by the Industry people concerned.
5. Identification of research issues to address on the formal methods and associated tooling to guarantee its transferability to Industry. This identification starts from Industry partners' needs that could not be addressed by the formal method or by its tools during the pilot project. This includes issues common to all sectors (e.g. tool must be more stable, editor must react faster to user inputs, tool must enable for parallel team work on same models) as well as identification of sector specific and even partner specific issues, for example, the formal method must handle real-time constraints or the tools must be able to generate code.
6. Identification of an advanced pilot project by each partner. This advanced pilot should be more challenging than the initial pilot project and may actually address a project that has not yet be executed by the company.
7. Industry partners work autonomously to model the advanced pilot.
8. Review and status of the overall transfer during the Federated Event during February 2012 in Paris.

With regards to timing, the initial intensive training (Step 1 above) was schedule at month 3 of project. This scheduling was necessary to identify the most appropriate time slot where nearly all Industry members could attend. a full-week course.

Each Industry partner evolved at their own pace on their respective initial pilot and mini-pilot. This intent was not to have all partners to follow the same pace since the evolution

could be radically different depending on people's familiarity with proof-based formal methods and the ease with which event-B could model the sector specific problem selected as initial pilot. Indeed, company with more background on formal methods tended to have more detailed and less ambiguous requirements document to start with. Thus, they could be much faster at modelling in event-B compared to partners with less detailed and potentially more ambiguous requirements document. Consequently, the period of work on the pilot and mini-pilot (steps 2-4) varied from 12 to 18 months.

Thanks to this initial learning phase on pilots and mini-pilots, Industry partners better understood what could or could not be modelled efficiently and elegantly using event-B. All felt ready to work on the advanced pilot after the initial learning phase.

Industry partners worked on their advanced pilot in the next two years of the DEPLOY project. Subsequently during the federated event in Paris, they all learn quite a bit from modelling their advanced project. After modelling the advanced pilots, DEPLOY Industry partner feel comfortable to claim to have reach complete autonomy with event-B.

Although an external organisation may not need to follow all the steps above, this approach provides an interesting path to draw from, in particular, with regards to its training and coaching pattern. All four Industry partners of the DEPLOY project were unanimously satisfied by this general training and learning approach where

1. An initial intensive training of 3 to 5 days accelerated the initial learning curve. Not only did the formalism is taught but the installation and use of associate tooling was covered.
2. On-demand coaching on a problem from their specific Industry helped them to gain confidence on their event-B modelling skills and their ability to handle the associated tooling.
3. Work incrementally on an initial pilot and then a more advanced pilot really helped Industry partners to become autonomous with the event-B formal method as well as other connected formalism such as ProB. Throughout this last incremental learning, Industry partners gain the impression to learn continuously and to finally reach full autonomy in the last year of DEPLOY. They also feel capable to assess their formal models and determine when alternative solutions need to be explored.

## *3.3 External factors advocating take-up*

Although different external factors may influence on the use of formal methods such as customer requiring the use of formal methods, competitors convincingly using formal methods, or law and regulation, at the moment, the most important influence comes from sector-specific standards. In this context, a manager should therefore want an answer to the following question:

- What is the position of standards regarding formal methods in my industry segment?

### 3.3.1 Stand of Standards on Formal Method

The landscape of standard includes both generic and domain specific standards. IEC61508 is the key generic safety standard and several domain specific standards specialise it, as shown in the following figure taken from [1]:

- IEC 61511 (published in 2003) addresses the industrial processes
- IEC 61513 (published in 2001) addresses the nuclear industry
- IEC 62061 (published in 2005) addresses the machine safety
- CENELEC/EN 50126/50128/50129 where (respectively published/updated in 1999/2001/2003) target the railway sector
- ISO 26262 (published in 2011) addresses the automotive sector

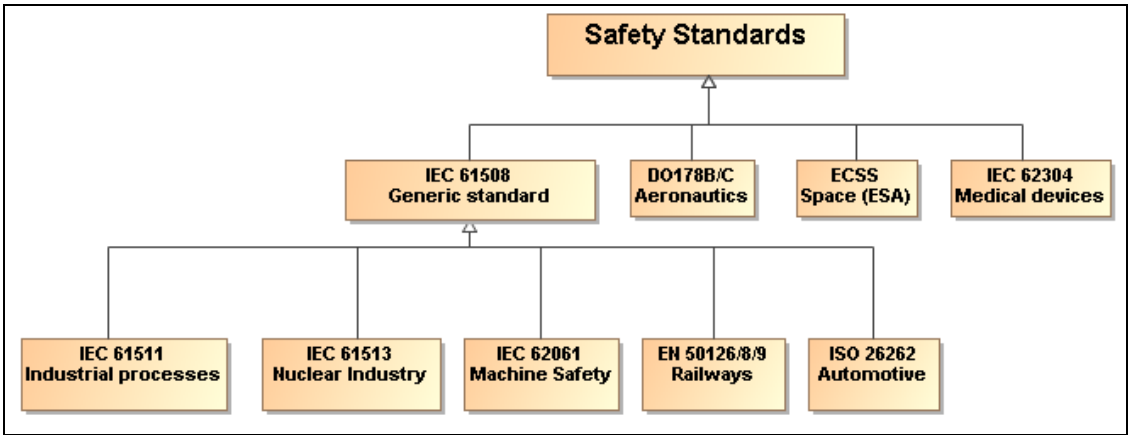


*Figure 1 Classification of safety standard*

In the aeronautic and space sector, there are standards not directly linked to IEC61508.

- DO-178 (1992) is for the aeronautic sector
- ECSS is for the space sector

In software related to medical devices, there are also specific standards, like IEC 62304 (2006).

Other generic standard may also have a more underlying role, for example ISO/IEC-12207 which specifies the software development lifecycle.

The above already shows a wide variety of standards. In addition to sector specific constraints, there are also a number of other reasons for such a variety of standards including historical reasons (uncoordinated work, national level) and market reasons (market protection) [2]. Moreover, those standards are also evolving hopefully to a better integration, so they are revised (the publication year is generally appended to their numerical identifier such as in EN 61508:2002) or more specific scheme (like the "B" in DO-178B).

Beyond this variety, standards exhibit common aspects:

- **Definition of safety assurance levels**: all standards have a risk oriented approach of their safety functions and classify them in a number of dependability classes, typically by specifying PFD (Probability of Failure on Demand) and RRF (Risk Reduction Factor) figures. This classification varies across the standards for example is IEC61508, the classification ranges from SIL1 least dependable to SIL4 most dependable. The DO-178B has a letter-base classification ranging from level "E" least dependable to level "A" most dependable.
- **Prescription level**: standards can be prescriptive (obligation to demonstrate compliance) or just give recommendations. However, in practice deviating from recommendations can be difficult as it may require non trivial work to motivate it and convince the certificating organisation used to the well-established recommended practices. This is an important factor that influences the adoption of formal methods since a certification authority may not appreciate to audit formal models and their associated proofs if they are not acquainted to them.
- **Scope**: it can address the software, the hardware or a complex system as a whole. It can also focus on specific part of the development process (like development lifecycle, quality management, safety assessment, system certification...). In the following table, inspired from [2], we give a classification of standards according to scope and domain.

<table>
<tr><th>Domain</th><th>System Certification</th><th>Development Process</th><th>Safety Assessment</th></tr>
<tr><td rowspan="2">Generic</td><td>ISO/IEC-15289</td><td>ISO/IEC-12207</td><td>IEC-61508</td></tr>
<tr><td colspan="3">CMMI</td></tr>
<tr><td>Automotive</td><td></td><td colspan="2">ISO-26262</td></tr>
<tr><td>Avionics</td><td colspan="3">DO178-B</td></tr>
<tr><td rowspan="3">Railway</td><td></td><td></td><td>IEC-50126</td></tr>
<tr><td></td><td>IEC-50128</td><td></td></tr>
<tr><td></td><td>IEC-50129</td><td></td></tr>
<tr><td>Space</td><td colspan="3">ECSS</td></tr>
</table>

## *Overview of standards position regarding Formal Methods*

### *Generic IEC61508*

Based on the SIL level, IEC 61508 classifies methodologies, techniques and activities as “not recommended”, “recommended”, “highly recommended”, etc. Among the “highly

recommended" techniques for SIL4 are inspection and reviewing, use of an independent test team and the use of formal methods.

To assure the required safety, reliability and correctness, a single "highly recommended" technique is not enough: it can only be reached using a carefully chosen combination of appropriate techniques. Reaching SIL4 in a project requires defining a dedicated system engineering process as described in [3].

***Railway sector***

The EN50128 guidelines, issued by the European Committee for Electrotechnical Standardization (CENELEC), address the development of "Software for Railway Control and Protection Systems", and constitute the main reference for railway signalling equipment manufacturers in Europe, with their use spreading to the other continents and to other sectors of the railway (and other safety-related) industry.

In EN-50128, Formal Methods/Proofs are explicitly identified as relevant technique/measure for software requirements specification, software architecture, software design, implementation, verification and testing and data preparation techniques. More precisely they are "recommended" for SIL levels 1 and 2 and "Highly Recommended" for SIL levels 3 and 4. Particular example of Formal Methods cited are: CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM, Z and B. In the on-going 2011 revision of the standard, additional constraints are put on tools, especially code/data generation tools with respect to specifications and evidence that the implementation complies with the specification by providing explicit traceability.

Despite this and success stories like METEOR, formal methods have not spread in the whole railway signalling industries, where much software is still written and tested in traditional ways (with testing effort usually summing up to more that 50% of the development effort). This lack of adoption is due to the investments needed to build up a formal methods culture, and to the high costs of commercial support tools. Moreover, equipment can conform to CENELEC even without applying formal methods [4].

However EN-50128 requires that bug detection and fixing activities be traced back not only in the production system but in the work products delivered at early phases of the development lifecycle, that is, the design and the requirements document. This causes higher costs to bug fixing. Consequently companies become interested in applying formal methods in the specification and design phases since this seems the only solution to shift back the effort to the design team and to identify potential problems before they are implemented in the executable system.

Although standard may mention formal methods, another barrier to their adoption lies in the competence of certification bodies with various formal modelling techniques in particular since most systems certified follow a test-based approach. The first time a certification body is confronted with documentation based on formal models, it will likely require some time of adaptation in particular, if artefacts traditionally required are not generated, for example, in their B development chain, Siemens does not perform any unit testing since they software units are correct by automated construction. Once established, new system can easily follow the same path of certification granted that the certification authority has learned from the past experience. Siemens has gone through

this process with the B-method, and it is now accepted by the certification bodies, so that the next projects have become easier to certify. In the specific market of metro lines, the B-method has even become the standard required by the market [5].

***Aeronautic sector***

DO-178B was published a while ago, in 1992. It does not recommend or propose a specific development process or methodology. The certification approach is to demonstrate compliance of the process and produced artefacts with a set of goals related to:

- software planning process
- development process
- artefact verification activities: ranging from requirements to design to code.
- integration testing
- verification of process
- configuration management
- certification

As mentioned above, DO178-B ranks the software category in 5 dependability classes: from A (most dependable: catastrophic effect) to E (least dependable: no effect).

The scope of the application of formal methods is the artefact verification activities. DO-178B certification can be achieved through a combination of three kinds of activities:

- reviews: relying on common software engineering practices: checklists, looking for specific kinds of defects/flaws (ambiguities, inconsistencies, incompleteness...)
- analysis: typically coverage analysis, data and control flow analysis
- testing activities: requirements based and with the need to demonstrate traceability

Related to Formal Methods, DO-178B categorise them as "alternative methods" because, at the time the document was produced (1992), they were evaluated to have an inadequate maturity level. However they can be used "as long as they can be demonstrated to address the goals of the standard, and their usage is adequately planned and described (...)" [6]

The wikipedia: DO-178C revision of the standard was published in January 2012. It is replacing DO-178B as the primary document by which the certification authorities such as FAA, EASA and Transport Canada will approve all commercial software-based aerospace systems. DO-178C is explicitly referring to formal methods to complement dynamic testing. They can be used selectively or as primary source of evidence.

***Automotive sector***

The sector specific standard is ISO 26262. It is based on the IEC50128 and was release in November 2011.

In ISO 26262, only a single development method may be highly recommended while several others may merely be recommended. Since current Automotive Industry practices are based on semi-formal methods, these methods have been assigned the highly recommended status. On the other hand, not to prevent the application of formal methods in the development of automotive products, Automotive Industry players involved in the conception of ISO26262 advocated recommending formal methods in the standard. It is therefore possible to be compliant to the current published ISO26262 standard when using formal methods to develop automotive products.

***Conclusion: general position of standards regarding Formal Methods and possible strategies***

Some standards recommend or even highly recommend formal methods but very few standards describe how to manage a formal development. It is therefore difficult to prove that the development process comply with the standards. Certification authorities have to be convinced about this issue.

As formal methods are less widespread and certification authorities are less familiar with them as compared to the classical development methods, there is more work to be done in comparison with other methods especially the first time around despite the fact that a stringer argument can usually be provided when formal methods are applied. However the investment might be worth the effort as shown by the Siemens case for B.

**References**

[1] Safety Blog (in French) - http://www.surete-fonctionnement.clearsy.com/2008/12/la-norme-cei-61508-et-ses-derives

[2] M. Bozzano, A. Willafiorita, Design and Safety Assessment of Critical Systems, Auerbach Publishers Inc, 2010.

[3] P. Kars. Formal Methods in the Design of s Storm Surge Barrier Control System. In Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems, Grzegorz Rozenberg and Frits W. Vaandrager (Eds.). Springer-Verlag, London, UK, 353-367, 1996.

[4] S. Bacherini, A. Fantechi, M. Tempestini, N. Zingoni, “A Story about Formal Methods Adoption by a Railway Signaling Manufacturer”, in Proc. FM 2006, Hamilton, Canada, August 2006, Lecture Notes in Computer Science, 4085, 1996.

[5] DEPLOY Deliverable D11.1, Measurement Methodology Guide, http://www.deploy-project.eu/pdf/d7-revised-final.pdf

[6] RTCA/DO-178B "Software Considerations in Airborne Systems and Equipment Certification", 1992

## 3.4 Understanding the impact of formal methods on the Software/System Development Process

To better understanding the impact of formal methods on the development process, DEPLOY Industry partners found the following topics interesting

- The impact on the quality of work products developed using formal methods,
- The capability to exploit formal models at various stages of the development process,
- The capability to perform reuse across development projects when formal methods are used, including reuse of formal and proven artefacts
- The capability to phase the learning of a formal method in an organisation and eventually to limit the scope of who must understand and become an expert in a formal method
- The capability to phase the migration to using a formal method incrementally (given the existence of products not initially developed using formal methods)

Several questions and answers on these different themes are addressed in the online FAQ (www.fm4industry.org). Below a set of three questions and answers of most interest to managers have been selected.

### 3.4.1 How do organizational procedures need to be adapted when formal methods are introduced?

Each company has its own role and task distribution scheme. In this answer, we consider the following roles and responsibilities:

- **High-Level Managers**. This level of management includes top-level managers of a production department or research and development department. Product and product line managers who are only concerned with the commercial aspects of a product are included in this management category. The responsibilities associated with the role of High-Level Managers are related to strategic decisions making and analysis of financial impact.

- **Project and QA Managers**. This group includes all managers who directly manage engineers, analysts, and QA practitioners. This group of managers may belong to production department or research and development department. The responsibility of this role is limited to determining the feasibility of a project and managing projects. These managers do not need to perform technical tasks but they often need to understand the technique, method and tools used by their teams to determine the feasibility and status of a project and decide if the resource allocation is appropriate to achieve a successful project.

- **Engineers and Analysts**. This role category represents people who apply formal engineering methods during system development projects.

- **QA Practitioners**. This role category represents people who do not necessarily apply a formalism but who need to understand it because they will have to review

and use documents containing the given formalism. QA practitioners are split from the role of Project and QA Manager because QA practitioners will ask different questions than Project and QA Managers. QA practitioners are in "do" actors versus the "supervise" activity of Project and QA Managers.

Below, we review how each of the role above will need to adapt his customary practices when formal methods are initially used on a project.

***For High-Level Managers***

The use of formal method dramatically shifts the workload of a project towards the analysis phase. Much more work is required in the analysis phase to develop formal models while much less is required at the testing phase. As such, managers need to:

- **Adapt their personal workload metrics** to accurately estimate the costs and delay for projects that use formal methods
- **Adapt their personal progress monitoring metrics** to monitor the proper progress of projects that use formal methods
- **Adapt their go/no-go procedure** to decide on the use (or not) of formal method in a project.

Estimating the cost factors in the context of formal methods requires some experience [1]. The factors involved in such go/no-go procedure are discussed in [2]. Basically, these include:

- **Obligation to use formal methods**, for instance, because it is requested by the customer
- **The development is internal, or on a shared-risk contract.** This is mainly to compensate for the possible low accuracy of cost and delay estimates.
- **The management is ready to face the cost shift** from late phase of the project to early phase. This can lead to an early red flagging of the project as being over schedule, and indirectly make it really of schedule due to increased reporting requests.

***For Project and QA Managers***

Project managers and QA managers need to properly design the development process and select the most adequate formal method to ensure that:

- **The claims that are proven by the formal method are relevant:** One can prove many things about a model; one should ensure that what is proven is useful to the project, and that all what should be proven is actually proven, or discharged to another validation method
- **The artefact on which the claim is proven is the most adequate one:** One can prove different things on different artefacts. For instance, one can prove some behavioural properties on abstract state machines or on the programme source

code. It is much easier to prove such claims on abstract state machines. This is a trade-off between cost (the earliest defects are detected the cheaper they are to correct, checking more abstract artefacts is simpler than more concrete ones) and level of assurance (the later the verification is performed, the fewer opportunities there are to introduce defects in the process)

- **The abstractions made in the model that is actually verified are valid:** Models introduce abstraction; one should ensure that the established proofs remain valid in the real world although they might be done on an abstract model.
- **The level of assurance that is delivered by the provers is adequate:** Not all formal methods deliver the same level of assurance. One should select the most appropriate formal method according to the current context.
- **The formal method is compliant with the targeted standard**. Some sectors require very high assurance, and might require proofs to be cross-checked by a redundantly with different provers, or to rely only on certified provers. Some provers of the DEPLOY project are certified for some CENELEC level of assurance.

***For Engineers and Analysts***

Engineers need to:

- **Develop formal artefacts:** This can be more-less time consuming depending on the considered artefact.
- **Formally define the claims that need to be proven:** These can typically be derived from requirements documents. Some formal methods natively include these claims, for instance if they are related to the proper use of programming language constructs like in software code verification tools à la Polyspace.
- **Prove the claims on the artefact:** depending on the considered formal methods, this can be more-less time consuming, depending on the level of guidance that is required by the tooling, the run time of the tooling, and, possibly, the intertwined development process where the model is gradually proven as it is developed.
- **Exploit the model in the next step of the development process:** When a formal model has been developed, and some claims proven on it, this artefact should be exploited in the development process. This can be through a translation process best carried by automated tools.

***For QA and Safety Engineers***

QA and safety engineers need to assess that

- **The claims that are proven by the formal method are relevant:** One can prove many things about a model; one should ensure that what is proven is useful to the project, and that all what should be proven is actually proven, or discharged to another validation method

- **The abstractions made in the model that is actually verified are valid:** Models introduce abstraction; one should ensure that the established proofs remain valid in the real world although they might be done on an abstract model.
- **The formal methods have been used in an adequate way.** For instance, one can prove any truth with a theorem prover if one includes an absurdity in the axioms.
- **The artefact on which claims have been proven is properly exploited in the downwards development process**
- **The model is easy to prove.** Some proof technology reach higher proof automation of performance if the model is developed according to some rules (avoid some type of constructs, avoid symmetries, etc.). Such verification can be performed by QA before the model is actually proven, to spare the time of engineers.

QA then amounts to checking that a formal model complies with the established guidelines. At Siemens, the above tasks are under the responsibility of the safety engineers. QA are in charge of verifying that the developed models match some reference good practices that have been synthesized into a set of guidelines. This verification is performed before proofs are made. Some of these good practices are related to the three aforementioned bullets while some others are aimed at reaching a high level of auto-proving. This is how Siemens reported to be internally organized.

**References**

[1] Calbaut, Mathieu Challenges in Applying Formal Methods -- An SME View. In: Dagstuhl seminar on Refinement Based Methods for the Construction of Dependable Systems, 14-18 Sept. 2009

[2] Donna C. Stidolph and James Whitehead, Managerial Issues for the Consideration and Use of Formal Methods, In Stefania Gnesi, Keijiro Araki, and Dino Mandrioli (eds.), FME 2003, International Symposium of Formal Methods Europe, 2003,8-14

### *3.4.2 What impact does the use of formal engineering methods have on the identification of issues at each phase of development cycle?*

Formal methods can apply at different stages of the development lifecycle to automate various types of verification that would otherwise be performed at later stages of product development lifecycle *or not performed at all*. As a result, it is generally admitted that formal engineering methods help to identify issues earlier in product development lifecycle unlike non-formal developments where issues are increasingly identified as the development lifecycle advances to reach usually a peak at testing time.

Therefore, by identifying errors earlier, formal methods forces one to resolve them earlier, for example, by removing ambiguities and incompleteness in the various work products produced as early as after requirement analysis or the design stage. In addition to removing errors, formal methods also help stakeholders to gain additional knowledge about the product or system being developed. For example, from a formal design, one

may learn about timing information in a complex system or from a formal analysis of source code, one may identify performance bottle neck, segments of dead code or deadlock freedom.

Below, various industrial case studies were reviewed to determine the impact of formal method usage on the identification of issues at various phases of the development lifecycle.

***Requirements Phase***

The use of formal methods at requirements analysis stage can be very beneficial to the whole development process as issues and error identify at this point are much cheaper to correct.

Requirements documents are usually insufficient for specifying formal models. In a non formal development approach, these insufficiencies can persist much longer in the development lifecycle. On the other hand, formal modelling forces requirement analysts to be very accurate and detailed in turn they ask very precise questions to customers to complete or correct problematic requirements. Indeed, given the wealth of information needed to develop formal models, requirement analysts learn to ask question that are often completely overlooked by non-formal development techniques. After a few years of experience with formal modelling, analysts become much better at asking directly the right question to obtain clear and complete requirement from the customer and they are also much faster at identifying requirements defects even before initiating a formal analysis.

Below is a short presentation on works performed during the DEPLOY project and other endeavours to show that the various facts mentioned above are verified at Bosch, Siemens, SSF as well as outside the DEPLOY consortium.

> *At Bosch, the substantial refactoring of a requirements document using the Problem Frames method helped to correct a number of problems such as inconsistency, incompleteness, and ambiguity. This is described in more details in a success story found in the website mentioned above.*
>
> *Since Siemens started using B to develop formal models of their software components, they have become much more thorough during requirement development and elicitation. In particular, they report that they now ask questions that were not even considered before when informal development techniques were used, for example, they now require an accurate ground topology of the metro system to reason accurately about braking distance to reach a halt.*
>
> *SSF initially attempted to model formally two different systems using Event-B, namely, the BepiColombo SIXS/MIXS on-board software and attitude and orbit control systems. The modelling exercise of BepiColombo SIXS/MIXS on-board software requirements in Event-B helped to identify ambiguity, incompleteness and redundancy that are commonly discovered later in the development cycle.*

Similar observations have also been made outside of DEPLOY:

> *In [1], [2], [3], it is explained that SCR (Software Cost Reduction) and its toolset were used to identify several serious issues during requirement analysis. SCR is*

*devoted to the proper identification of requirements through simulation and validation of state machine by end users. SCR is furthermore able to perform simple checks on the formal models such as determinism and forms of logical completeness. This method is still in use nowadays and has matured for more than 15 years.*

*In [4], two case studies explored the impact of two formal techniques for specifying and checking requirements. In one case study, SRI's PVS specification language was used and Stanford's Murphy finite-state verification system was used in the other. Among various benefits of using these two formal requirement analysis, the authors compares the issues identified when using formal requirement analysis and when only performing non-formal reviews. They discovered that the use of formal requirement analysis help to identified 7 major issues while non-formal approach only discovered 1 error; concerning minor issues, 23 were discovered using formal method and 3 only through non formal checks.*

*In [6], The NewCoRe project that ran over a two year period in the early 90's present the use of formal method to validate communication protocol in the Telecom sector. A specification of 7,500 lines of (non-commented) SDL code was written and about 150 correctness properties were formally specified and verified for the SDL model. As a result, a total of 112 serious design errors were detected in the design requirements.*

***Design Phase***

Using formal methods at the requirement level aims to ensure that the functionality of the product are well understood and described, and that all angles of that functionality have been analyzed and specified in details in the requirements document. Although requirements document make general statement regarding non-functional requirements as well as describe constraints from the environment, requirement analysis cannot really guarantee non-functional behaviours because requirements do not usually specify the complete architecture of a proposed solution. Consequently, formal specifications of certain non-functional behaviour are left for later stages of the development lifecycle.

The most important observation from the field concerning issues identified at design time comes from Siemens. They currently use the B method to design their software component for metro and train systems. In their design, all safety conditions of their formal models are proved to be satisfied. Thus, a system implementation that matches the design will never encounter safety problem (within the scope of the considered requirements). Clearly, when they encounter a problem with a proof, this forces them the redesign their formal models until all safety conditions are guaranteed. Certain proofs can take one to two months to be verified by humans. Thanks to Pro-B, a model checker and animator build for B and Event-B models, Siemens can now check if a counter example exists in matter of seconds thus avoid wasting time trying to proofs non-provable facts. We can therefore argue that in the Siemens case, formal modelling not only helps to build safe transport systems but also do so at a much improved productivity rate.

Another observation relates to the use of formal vs informal techniques to verify the correctness of models. At SSF, two teams performed the verification of *the BepiColombo SIXS/MIXS on-board software and attitude and orbit control systems.* The first team used informal review techniques while the second team created formal models of the system

using event-B to prove the correctness of the system.

> *A SSF team modelled the architecture of two different systems using Event-B, namely, the BepiColombo SIXS/MIXS on-board software and attitude and orbit control systems. The team who model in event-B identified several issues linked to ambiguity, incompleteness and redundancy in requirements. Although these associated observations were also identified by the other team not using formal method (mostly based on algorithm inspections), the power of Event-B models raised the confidence of design analysts much more on the correctness of the proposed system architecture and algorithms. In conclusion, applying informal verification techniques at design time is very helpful however it is hard to determine when a sufficient effort has been spent. On the other hand, when using formal methods, the sufficiency of design verification is limited to determining if proven properties cover an wide spectrum of the real world situations.*

Another important observation concerns to the impact of formal design on later stages of the development cycle. In particular, SSF noticed that using the Event-B formalism for modelling attitude and orbit control systems helped to create simpler more modular system architecture easier to understand for system developers. Consequently, they believe that developers will introduce fewer errors at the implementation stage.

***Implementation & Debugging Phase***

Siemens has implemented a complete formal chain from design to code by using a code generator (recognized for SIL 4 certification). B specifications proven correct are fed to the code generator that generates the full code. No edition to this code is performed and given the correctness of the code generator, no issues are identified during the testing phase, which can therefore be eliminated.

A more traditional approach to applying formal methods on source code is Source code analysis, which has proven effective in Industry in the last decade. In such case, the development approach to obtain source code may be informal and then tools for model checking or performing abstract interpretation are use to prove certain properties of the source code program. Notably, the Polyspace tool is a well-known example that performs static analysis of C, C++ or ADA code through abstract interpretation, thus relying on automated algorithm [5]. It has been extensively deployed in Space and avionics sectors for instance. In [7], the authors explain how Astrée has been used to prove certain properties of avionics systems.

***Testing Phase***

At Siemens, code is generated from B specifications proven correct. Hence, they needn't perform any unit testing as no issue would be identified except if errors exited in the code generator or complier. Both of those used at Siemens have been certified for use at SIL4. However, given the habits of most customers to participate to acceptance test session to sign the final acceptance record, Siemens still performs acceptance test to validate the overall functioning of the software. In other words, given the very thorough design specification induced by the use of the B formal method, all requirement issues are identified during the design phase. Acceptance testing is only limited to performing a demonstration of the system to the customer using animation for example.

**References**

[1] Constance L. Heitmeyer, "Formal Methods for Specifying, Validating, and Verifying Requirements", J. UCS, 5:13, pp 607-618, 2007.

[2] S. M. Easterbrook, R. R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton, "Experiences using lightweight formal methods for requirements modeling." IEEE Trans. Software Eng., vol. 24, no. 1, pp. 4–14, 1998.

[3] S. Miller, "Specifying the mode logic of a flight guidance system in CoRE and SCR," in Proceedings of the 9th 2nd Workshop on Formal Methods in Software Practice (FMSP'98), 1998.

[4] Crow, J., Di Vito, B.L.: Formalizing Space Shuttle software requirements: Four case studies. ACM Transactions on Software Engineering and Methodology 7 (1998) 296–332

[5] Polyspace product for embedded software verification http://www.mathworks.com/products/polyspace

[6] G. Holzmann., *The theory and practice of a formal method: NewCoRe* In *Proceedings of the IFIP World Computer Congress*, volume I, pages 35-44, Hamburg, Germany, August 1994. North-Holland Publ., Amsterdam, The Netherlands.

[7] Jean Souyris and David Delmas, *Experimental Assessment of Astrée on Safety-Critical Avionics Software* in Proc. Int. Conf. Computer Safety, Reliability, and Security, SAFECOMP 2007, Francesca Saglietti and Norbert Oster (Eds.), Nuremberg, Germany, September 18—21, 2007, Volume 4680 of Lecture Notes in Computer Science, pp. 479—490, © Springer, Berlin.

### *3.4.3 Can the use of formal engineering methods help in the design of tests?*

This short answer is clearly YES.

Deploying a full formal chain from design to code as done at Siemens implies a tremendous shift. In the Siemens case, this shift was imposed by their customer, the RATP (Paris organisation responsible for the metro system). Thus, it was their only choice if they were to continue to conduct business with RATP.

In most other cases, reaching such a level of formality is not an option, at least not initially. In such cases, it is more customary to produce formal models to prove that the designed architecture is adequate to meet certain properties. Subsequently, these formal models are handed to the development team that uses an informal development approach. It is however possible to harness the power of the formal model to generate test cases and potentially a test oracle.

Overall, there are at least four ways the design of test can be impacted by the use of formal methods:

1. Certain kinds of tests can become unnecessary if a formal method chain results in a proven implementation;

2. Test cases can be automatically derived from formal models with assurance of some type of coverage;

3. In a manual design process, deriving tests from formal models can be more systematic than from informal documents;

4. The use of formal methods results in requirements that are more precise. An important side effect of better requirements that they facilitate to develop better test cases.

**1. Removing the need for specific kinds of tests**

When formal methods are in use, some tests might be dropped, or partially dropped. For instance, if one validates that a communication protocol is deadlock-free through formal methods, one will not focus the testing of the implementation of this protocol on a search for deadlock. Rather, one can validate that the implementation complies with the verified model, and test properties of the protocol can be omitted.

As a concrete case, Siemens does not perform any unit testing of the software code that is developed through the B formal method, as the B models are proven correct by construction, and the code is generated from these B models through a certified code generation process relying on automated code generation tools. Rather, they focus the testing on the integration phase, to validate e.g. the requirements and the assumption on the domain. Of course, switching from a test-based validation to a formal method-based validation must be done in accordance with the targeted norms or standards.

**2. Model-based testing**

One can derive test cases from formal models. This is known as "model-based testing" [1, 2, 3, 4, 5]. Model-based testing automates the detailed design of the test cases and the generation of the traceability matrix [1]. More precisely, instead of manually writing several test cases, the test designer writes an abstract model of the system under test, and then the model-based testing tool generates a set of test cases from that model. The tests are generated so as to enforce some completeness property on the model.

For state machine, one can ensure the following test completeness properties:

- Covering each state of the model
- Covering each transition of the model
- Covering each transition pair of the system (provided they can all be taken at some point)
- Covering each simple acyclic path

Model-based testing has two main advantages:

- First, the design time of test cases is reduced.
- Second, one can generate a variety of test suites from the same model by using different test selection criteria.

Existing tools for model-based test generation include [6, 7, 8, 9, 10, 11]. Formal models from which test cases are generated can be any transition system including state machines, possibly incorporating some executable software code, or formal models such as B or Event-B. In the DEPLOY project, SAP showed model-based testing was feasible based on Event-B and ProB tools [12].

**3. Manual design from formal models**

Once high quality specifications or models are available, for instance as the result of a formal development process, one can identify test cases in an easier way from these models even on a manual basis. For instance, identifying a test case can be done more easily if one aims at following a given path in an explicitly modelled state machine. This is more-less the process that is automated by model-based testing tools.

Besides this, team working is made easier because engineers can easily communicate based on precise models, and collaborate to develop test suites.

**4. Better requirements equals better test cases**

Designing formal models will force designers to explore thoroughly the meaning of requirements. In most cases, formal methods forces designers to iterate with the customer to understand fully the system under design. This will then generate update to requirements document to make it more precise, more concise (remove of duplicate statements), less ambiguous. Consequently, it forces one to deliver higher quality requirements documents. Thus even if the generation of test cases remains manual, high quality requirement enables one to more efficiently write relevant test cases.

**References**

[1] Mark Utting and Bruno Legeard, Practical Model-Based Testing: A Tools Approach, Morgan-Kaufmann 2006

[2] Jacek Czerwonka, www.pairwise.org, Pairwise Testing; Combinatorial Test Case Generation, Last updated: December 2008

[3] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, ‘‘Test selection based on finite state models,’’ IEEE Trans. on Software Eng., vol. 17, pp. 591-603, 1991

[4] G. Gonenc, ‘‘A Method for the design of fault detection experiments,’’ IEEE Trans. Computers, vol. C-19, pp. 551-558, 1980.

[5] Jonathan Jacky, Margus Veanes, Colin Campbell, Wolfram Schulte, Model-Based Software Testing and Analysis with C#, Cambridge University Press

[6] http://www.all4tec.net

[7] http://www.codeplex.com/NModel

[8] http://www.smartesting.com

[9] http://research.microsoft.com/en-us/projects/SpecExplorer

[10] http://www.conformiq.com

[11] http://www.trusted-labs.com

[12] http://www.stups.uni-duesseldorf.de/ProB

## 3.5 Formal Method Tools and Quality of Support

The online evidence FAQ identifies the various platform on which different formal method tools run. However, tools evolve hence what is true for a tool today changes in tomorrow. Therefore, this report prefers to focus on generic questions that a manager should ask regarding formal method tools and tool support. This is even more important as many formal method tools are the results of research projects that have released tools under open source licences. Many managers are still cautious when it comes to open source partly, because it is a new distribution model with several unknown factors and also because they are not clear on how to assess the quality of these tools and tool support from tool providers. Although generic, the answers to the two questions below hope to relieve part of the worry by guiding manager to asking the important questions, in particular when open source is considered.

### 3.5.1 What are important questions to ask about formal method tools to determine their readiness for Industry?

During DEPLOY, the four Industry partners as well as other associated partners have used and given feedback on the various formal method tools used. Based on their experience they found the following point most important [1].

- *Are there guarantees of long term Tool availability and Support?*

Industry projects might last tens of years between the development and the decommissioning of a system. It is crucial for Industry to ensure proper support throughout the complete project lifetime including its retirement. Tools can be distributed under Open Source or Proprietary Licenses. Each model comes with its own risk to disappear (bankruptcy for proprietary code vs. community disappearance for Open Source). Given the niche market, securing the support is not a trivial task (e.g. escrow for proprietary code, direct community involvement or support for Open Source).

- *Is the Tool reliable?*

Closed source reliability is a matter of trust which can be provided by a certification scheme for example. Concerns have been raised about Open Source tools capability to achieve high reliability [2]. However the large number of industrial strength tools available nowadays tends to prove the contrary: e.g. PVS, nuSMV, and several others. Some reasons are related to the potential of massive peer review and at the design level, better defined interfaces and careful designs required for a distributed development. Furthermore, extensive test suites are often available for such Open Source tools.

- *Is the Tool scalable?*

The ability to scale up depends on different factors. Tool-induced limitations may be due

to the underlying formal technology, implementations problems (e.g. some bottleneck in a processing chain) or simply usability (e.g. limitation to manage large pieces of models). To assess scalability, references, feedback and reviews provide initial information useful to directly rule out inadequate tools for Industry. A second step is to challenge the tool on realistic case study in various Industry sectors since the way models are build can also impact on the ability to scale up. Open Source tools might have higher risk of not scaling up, especially if they are still at the R&D stage. However there are also highly scalable Open Source tools in the area of formal methods e.g. SPIN and nuSMV are scalable model-checkers, ACL2 and Isabelle are scalable theorem provers.

- *Is the Tool usable?*

It is important that tools ease various tasks when building or modifying a model, carrying out validation and verification activities, working in team, etc. Commercial tools generally have better usability because a special attention is devoted to this aspect while Open Source tools tend to focus more on the core functionality and efficiency, with sometimes only a command line interface.

- *Does the Tool integrate well in Industry tool chains?*

The ability to integrate into existing industrial tool chains is fundamental. This requires the existence of well-documented data format, availability of APIs/binaries on specific OS's/integration with popular tool platforms. This is an area where Open Source usually outperforms proprietary tools. Furthermore, Open Source often adopt open standard data format. On the other hand, heighten competition frequently pushes proprietary tools to keep internal data format hidden to force vendor lock-in.

- *What is the impact of tools on certifying an Industry product?*

Using a formal tool in the design flow (i.e., at design time) might have an impact on the certification process, especially if the tool is generating work products considered for review by certification authorities such as source code for systems requiring high integrity levels. Evidence of correctness of the output produced by those tools has to be provided by various means: redundant implementation, extensive test coverage, specific verification activities. As supporting success story for this, the ProB tool used by Siemens and developed by the University of Düsseldorf is undergoing a qualification for the railways EN-50128 standard.

**References**

[1] Christophe Ponsard, Jean-Christophe Deprez, Renaud De Landtsheer, *Is my Formal Method Tool Ready for the Industry?*, Proceedings of the 11th International Workshop on Automated Verification of Critical Systems (AVoCS 2011).

[2] D. Craigen. *Formal Methods Adoption: What's Working, What's Not!* In Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking. Pp. 77–91. Springer-Verlag, London, UK, 1999.

### 3.5.2 What aspects of tool supports are important for formal method tools released under open source licences?

An enterprise may decide to take an active part in the development of a formal method and its associated tooling however in most cases it only wants to build an expertise on a selected formalism and to acquire the software tooling associated to the select formalism. In case of problems, the enterprise also wants to obtain support. Supports may take different forms such as help in installing tools, training in using tools and learning the formalism, obtain correction or work around in case of problems with a tool, and in case of a major blockage when modelling with the formalism, an enterprise may also want to hire an external expert to help with the real world formal modelling exercise.

Managers are acquainted with the acquisition process of proprietary tools on the other hand they are usually not well-versed on how to acquire open source software. Several open source assessment methods have emerged over the last decade, some from European projects such as QualOSS (http://www.qualoss.eu) or Qualipso (http://www.qualipso.eu). In addition, QSOS (http://qsos.org) and OpenBRR (http://www.openbrr.org) are two other open source assessment methods. All of them follow a similar assessment approach where quality indicators are inferred from lower level measurements related to software as well as to the community, its development process and other information regarding licences and service providers.

A sample assessment result based on the QualOSS standard-assessment method applied on Rodin tool performed in January 2009 is available on the evidence FAQ at the following URL: http://www.fm4industry.org/index.php/TOOL-HM-1.

Alternatively, http://ohloh.net provides measurements for many open source software. Although not performing a qualitative assessment, Ohloh provides a initial set of information, for example, to determine in a matter of seconds how many commits have been made recently, how many committers have participated and to which degree. It may therefore be of interest to a manger to take a quick glance at Ohloh data before considering a more thorough assessment. Unfortunately, not all formal method tool are found in Ohloh, notably, Rodin is referenced but has not measurement data available.

# 4. Conclusion

This HOW-TO guide for managers proposes an initial set of questions/answers to help better understand the impacts of formal method adoption in Industry. This guide is supplemented by a wiki style website open for collaboration.

In addition to having more questions and answers as well as cross-references, the online FAQ of evidence also include a set of success stories from the DEPLOY project. Unlike Q/As that address cross-industry concerns, success stories are pieces of information specific to a transfer action at a single DEPLOY partner. Although context-specific, DEPLOY success stories may also provide inspiring material for managers to learn about formal methods and hopefully to make their initial move to experimenting with formal methods with their development team.