How to Evaluate the Suitability of a Formal Method for Industrial Deployment? A Survey Technical Report SCCH-TR-1603

Felix Kossak, Atif Mashkoor
Software Competence Center Hagenberg GmbH (SCCH)
Hagenberg, Austria
firstname.lastname@scch.at

March 9, 2016

Abstract

Despite extensive evangelizing and demonstration of several success stories in safety-critical applications, formal methods are still not widely practiced in contemporary systems and software engineering. One of the main reasons for this situation is the absence of systematic guidelines and evaluation criteria that help software practitioners choose the right formal method for the problem at hand. In this article, we present a comprehensive set of criteria for evaluating and comparing different formal methods, based on a systematic literature review and decade-long personal experience with the application of formal methods in industrial projects. We argue that besides technical grounds (e.g., modeling capabilities and supported development phases), formal methods should also be evaluated from social and industrial perspectives. We also evaluate several state-based formal methods on the stipulated criteria.

1 Introduction

Despite many years of advocacy, numerous success stories in safety-critical systems, and the availability of various "easy to use" methods and tools, the application of formal methods is still marginal in mainstream software development. Several factors can be held accountable for this result. One of them is that no proper guidelines are available at the disposal of software practitioners which navigate them through the intricate process of choosing a formal method suitable for their problem domain.

Different formal methods are generally suitable for different kinds of software projects, domains, and social and economic settings. For instance, the development of safety-critical systems will require elaborate evidence for compliance with safety requirements and standards, while in other projects, budget and time restrictions will not allow for expensive verification efforts. As another example, it makes a difference whether mostly mathematicians or engineers specially trained for a particular domain are involved in a project, and will also be available for maintenance later on, or whether the methods used must be suitable for ordinary software developers having little familiarity with complex mathematical and logical structures.

Several studies have already been published where individual formal methods are compared. However, as we will detail in Section 2, many of these studies are either outdated or concentrate on limited aspects of application of formal methods (often either merely technical criteria of predominantly academic interest or tailored to a particular domain of application). Additionally, none has presented general guidelines/evaluation criteria which may help software practitioners in choosing the right formal method for their problem at hand.

In this article, we present a comprehensive list of criteria for comparison of formal methods with respect to general industrial interest, drawn from a structured literature review as well as personal experience that stems from the application of formal methods on several industrial and academic projects, for example, hemodialysis machines [MBDT15],[Mas15], aircraft landing gear system [Kos14], machine control systems [MHB13], transportation systems [MJ11], [MJ10], platooning systems [MJ15], and business process modeling [KIG+15]. In contrast to many other publications, we include a wide range of criteria which we deem crucial for a wider adoption of formal methods in the industry.

The main goal of this study is to provide guidelines to software practitioners to help them choose a particular formal method, or maybe a small set of methods, for a particular software (or software-hardware co-development) project. Thereby the focus is laid on industrial projects, including large-scale projects. The motivation behind this goal is to provide necessary means to help propagate the use of formal methods in day-to-day systems and software engineering.

In this article, we attempt to answer the following three research questions: 1) What criteria are useful in order to select a particular formal method for a particular setting? 2) Why are the criteria important for the evaluation of a particular formal method? and 3) How do particular formal methods fare with respect to these criteria?

This article is structured as follows: First we present our research approach and the literature reviewed in Section 2. Then in Section 3, we present a structured list of criteria for evaluating formal methods

and explain why these criteria are important. In Section 4, we compare several state-based formal methods with respect to these criteria. In Subsection 4.10, we further present a simplified table designed to enable project-specific assessments of methods. The paper is concluded in section 5, which provides a summary and an outlook for the need of future research.

2 Approach and Literature reviewed

2.1 The research approach

In this paper, we answer the following three research questions:

1. What criteria are useful in order to select a particular formal method for a particular setting?
2. Why are the criteria important for the evaluation of a particular formal method?
3. How do particular formal methods fare with respect to these criteria?

Our research approach is based on a structured literature review complemented by our own experiences with several formal methods. We limited the literature research to an Internet search of articles with the following search strings:

- "formal methods" AND "evaluation criteria"
- "formal methods" AND "comparison"
- "formal methods" AND "state of the art"
- "formal methods" AND "literature review"

We stopped after seven pages of search results, after which the relevance of results dropped markedly. We further included the literature which we were already aware of.

The literature research showed that several comparisons between different classical formal methods were conducted in the 1990s and around 2000. Recently, more comparisons were made in special settings, typically in the context of university courses. We noted a recent surge in formal method-related tools which can be integrated in traditional development platforms. These are typically static checkers or model checking tools that only partially cover specification and model-based verification against custom safety properties. Most existing studies compare only a few methods, often only two or three. Evaluation criteria vary widely, revealing different possible viewpoints.

Evaluations of formal methods from the 1990s must certainly be considered outdated, for much has changed since, in particular with

respect to tool support, the amount of practical experience, and how widespread a method is used. This does not leave much material for a concrete evaluation of particular methods. Still, older publications can yield interesting contributions to the criteria by which formal methods should be evaluated (sometimes presented as wishes). Often the focus is on a purely academic viewpoint in this respect, but not always.

2.2 Literature reviewed

We now present the literature (in order of relevance) which we found interesting for the current study. (We do not include sources which only yielded information on a single method in this place.)

Information on concrete evaluations within the industry appears to be scarce, though we assume that such evaluations happen. A notable exception is a recent paper by Chris Newcombe, "Why Amazon Chose TLA+" [New14], though it largely only describes experience with TLA+ and, to a lesser extent, Alloy [Jac06] and Microsoft VCC [CDH+09]. Newcombe's criteria are drawn from the very demanding domain of cloud infrastructure services. Key demands for those services are a high level of distribution, high performance, and high availability.

A position paper from Joseph Sifakis [Sif97] also discusses industry-centric evaluation criteria and provided useful input for the list presented in this paper. Also Sifakis discusses, amongst others, the crucial point of usability and human factors in general.

The papers from Ardis et al. [ACJ+96] and Knight et al. [KDGN97] also provide frameworks for the evaluation of formal specification languages. They first present criteria and then evaluate several formal languages. The latter also present the perspective of developers, engineers, and computer scientists on these languages.

In "Formal Methods: Practice and Experience," Jim Woodcock et al. [WLBF09] contribute an overview of historical experiences with formal methods, in particular from industrial projects. We could extract several important criteria from this paper, in particular with regard to tool support.

Thomas McGibbon [McG97] discusses different evaluation criteria from a viewpoint of the U.S. Department of Defense, including more detailed requirements for tools.

Several evaluation criteria can be extracted from the seminal papers by Edmund Clarke et al. [CWA+96] and Bowen et al. [BH95, BH06]. The former present the state of the art and future directions of formal methods and the latter present some guidelines to help propagate the use of formal methods in industry. Those criteria can basically all be found in later sources as well.

X. Liu et al. [LYZ97] list a number of evaluation criteria and compare a great number of methods. Amongst others, the authors bring in the additional criterion of applicability in re-engineering, in particular in reverse engineering and restructuring. Although they are primarily concerned with support for re-engineering, this paper is also of general interest; it includes interesting characterizations of many different methods and their state at the time, though unfortunately much of this information is now (potentially) outdated.

Richard Banach, in "Model Based Refinement and the Tools of Tomorrow" [Ban08], compares B [Abr96], Event-B [Abr10], Z [Spi89], and the ASM method [BS03] from a mathematical/technical point of view. His paper contains the only direct comparison one can find of the ASM method with other methods.

Also a book by John D. Gannon, Marvin V. Zelkowitz, and James M. Purtilo, entitled *Software Specification: A Comparison of Formal Methods* [GZP94], focuses on mathematical issues; it discusses only VDM [Jon90] as a formal method in a closer sense, together with temporal logic in general as well as "risk assessment."

Also Arvinder Kaur et al., in "Analysis of Three Formal Methods - Z, B and VDM" [KGS12], stress mathematical and modeling issues, but they also mention, e.g., tool support, code generation, and testing.

In the book *Software Specification Methods* (ed. by Frappier and Habrias) [FH06], many different methods are introduced by means of a case study. In the last chapter [FHP06], some (but not all) of the methods are compared in tables. The (purely qualitative) criteria include some which we chose not to adopt here, including graphical representation, object-orientated concepts, use of variables, and event inhibition.

In a master's thesis, Marian Rainer-Harbach [RH11] compares several different proving tools for software verification, but not any comprehensive method which could support other project phases and aspects.

Maurice H. ter Beek et al. [tBBG07] wrote a paper on "Formal Methods for Service Composition," dealing with a very narrow field of application. They compare only model checkers, Petri nets, and process algebras.

A paper by Giovanna Dondossola [Don98] specializes on the application domain of safety-critical knowledge-based components and on the method TRIO [GMM90]. Towards the end, it also offers a comparison of different formal methods, including VDM and Z; however, the criteria used there are described only very coarsely, so we could not extract much extra information for our purposes.

A technical report by Milica Barjaktarovic [Bar98] names requirements for formal methods, and in particular industrial requirements, throughout the text; most of those requirements are also found in other

sources, but this paper provides a good confirmation.

Bicarregui and Matthews [BM95] compare VDM and B based on experience gained in two industrial projects, with a focus on proving.

From an article by Pandey and Batra [PB13], we obtained useful assessments of Z and VDM, in particular.

3 Criteria for Evaluating Formal Methods

Now we will present a structured list of criteria which we deem relevant for assessing and comparing formal methods for their usefulness in concrete industrial projects, depending on the concrete settings of a project. We first give an overview of the criteria we found and deemed relevant before describing each of them in more detail and explaining their significance. Please note that the classification of certain criteria under a particular category may be cross-cutting and overlapping to some degree. This is by choice as this makes each category an independent unit of analysis that can also be taken into consideration in isolation for concentration on a particular class of criteria.

3.1 Overview

We found five categories of criteria relevant for industrial projects:

1. **Modeling criteria:** What possibilities and scope for modeling and refinement does the method offer?
   - Support for composition/de-composition
   - Support for abstraction/refinement and what notion of refinement is employed
   - Support for parallelism/concurrency/distribution
   - Support for non-determinism
   - The possibility to express global system properties of correctness
   - Support for the modeling of time and performance properties
   - Expressibility of various special (domain-specific) concepts (e.g., differential equations or user interface aspects)
   - The possibility to express rich concepts *easily*
2. **Supported development phases:** Which phases of a software (and/or hardware) development project can be supported (and how)?
   - Specification
   - Validation
   - Verification

- Bug diagnosis
- Architecture & design
- Coding/code generation
- Testing
- Maintenance
- Reverse engineering

3. **Technical criteria:** What tools are available, and how do the method and the tools interact with other development requirements from a technical point of view?
   - Overall tool support
   - Commercial support for tools
   - Customizability of tools
   - Traceability of requirements and during refinement/code transformation
   - Support for change management (How much stability of the initial specification is presupposed? What about maintenance of the finished product?)
   - Effect of the method on development time (for specification, validation, verification, etc.)
   - Efficiency of generated code (Can the generated code be used as it is? How much manual tweaking is necessary?)
   - Efficiency of code generation (How fast does code generation work? What does a small change in the model mean for subsequent code re-generation?)
   - Interoperability with other methods and/or other tools
   - Integration of methodology and tools with the usual development methods and tools (IDEs)
4. **Usability:** How easily can people with different backgrounds and expertise handle the method and its results? How can people collaborate when using the method?
   - Learning curve (How fast can one learn the method from scratch? What prior expertise is required?)
   - General understandability (Is the model understandable for non-experts? Can the model be made accessible via visualization/animation?)
   - Available documentation (including case studies)
   - Support for collaboration
5. **Industrial applicability:** How well can the method be used in potentially large and complex industrial projects, and what industrial experience is there so far?

- Support for industrial deployment
- Scalability
- Amount of (industrial) experience
- Is specialized staff required, and if so, to what extent?
- Standardization
- Availability and licensing of method and tools

3.2 Modeling criteria

Modeling criteria concern the scope of systems and requirements which can be modeled and formalized as well as the ease, e.g., the succinctness, with which such modeling is possible.

The first criterion is *support for composition and de-composition.* De-composition is important for any domain of application when it comes to "larger-than-toy" systems; without de-composition, large models cannot effectively be overviewed and handled. De-composition is also of great value in correctness proofs. Composition is not only important as a necessary ingredient to de-composition (gluing de-composed part models together), but also for re-use. Amongst others, (de-) composition is explicitly mentioned as a criterion by [Sif97], [ACJ+96], [McG97], [CWA+96], [LYZ97], [Ban08], [KGS12]. A related issue is reuse, which should be supported by appropriate possibilities for parametrization in de-composition. Reuse is explicitly mentioned by [CWA+96], [BH95].

Next is *support for refinement*, that is, for building a series of models for the same system with increasing depth of detail or, for reverse engineering, with increasing abstractness. If refinement is supported, then also the design phase can be supported by the method. Ideally, it can even become possible to refine a model up to the level of detail required for implementation, or actually right down to programming code. Going into the other direction, reverse engineering can be supported. Refinement can further be employed to make a model executable for validation and testing. Refinement is mentioned as a criterion by [New14], [Sif97], [McG97], [LYZ97], [Ban08]. Banach [Ban08] also investigates methods with respect to what notion of refinement they employ, which is relevant for correctness properties. Also [CWA+96] mention refinement explicitly, as well as "evolutionary development."

Support for modeling *parallelism, concurrency, and distribution* is essential for a wide range of real-life applications. We can distinguish between synchronous parallelism and asynchronous concurrency; the latter can be further complicated by arbitrary distribution of resources. While the modeling of (synchronous) parallel systems is well understood, modeling of (asynchronous) concurrent systems is

still subject to research. Yet the latter is highly relevant; a paradigm example of highly distributed concurrent systems is cloud services. *Parallelism* is mentioned by [Don98]; *concurrency* is mentioned by [New14], [McG97], [LYZ97], [KGS12]; *distributed systems* are mentioned by [New14], [Sif97], [LYZ97].

Support for non-determinism is very useful for keeping models abstract. For specification or high-level design, many details needed to make a model deterministically executable are not only unnecessary, but actually unwanted. Overspecification distracts and impairs overview [Mey85], and for many details, it is better to leave them to implementers to decide. For execution of abstract models for the purpose of validation, tools offer ways to randomly fill the gaps left by non-determinism in the model, so non-determinism is not a disadvantage in this respect. Out of the literature reviewed for this work, non-determinism is mentioned as a criterion only by [LYZ97]; however, it is implemented in several methods and motivated in the respective method-specific literature, e.g., [BS03].

The possibility to express global properties of system correctness is necessary to be able to prove respective requirements such as safety and temporal constraints (termination, deadlock freeness, fairness). [Sif97] notes this criterion explicitly. An important class of global properties are reliability properties, which are explicitly mentioned by [McG97], [LYZ97]; security policies are mentioned by [CWA+96].

Support for modeling time must regard sparse and dense models for time separately (see [LYZ97]). The former is required for general model checking, the latter for modeling real-time properties (including performance) and respective model checking. The importance of having an explicit notion of time in a modeling language is stressed by [New14], [LYZ97], [Don98]. Performance properties concern the complexity of algorithms both with respect to time and with respect to memory use. Modeling of performance properties and/or real-time constraints is explicitly mentioned by [New14], [Sif97], [McG97], [CWA+96], [LYZ97].

Many domains of application require that *special concepts* be easily expressed in a modeling language. One important case is hybrid systems which require the modeling of both discrete and continuous state changes – see, e.g., the work by Richard Banach et al. [BZSH11]. Hybrid systems are also mentioned in [CWA+96] and [AFPMdS11, Chapter 2]. Another example is the modeling of probability, as desired by [CWA+96], [LYZ97]. [LYZ97] also evaluate methods with respect to their ability to model different communication concepts (especially synchronous and asynchronous communication; see also *concurrency* above, although Liu et al. [LYZ97] separate these issues). Other examples include the modeling of usability properties [McG97] or of user interface aspects, in particular in safety-critical environments such as interfaces for pilots or air controllers [McG97], but also queuing the-

ory [CWA+96]. [AFPMdS11, Chapter 2] note that (often continuous) properties of physical systems frequently need to be modeled in the context of real-time systems. [Bar98] mentions the criterion of "domain specific notations for niche markets" in general. The general criterion listed is, of course, domain-specific, but, e.g., existing libraries of re-usable concepts and related proofs may play an important role here.

We can generally expect a desire in industry to "be able to capture rich concepts without tedious workarounds" [New14], which expresses the last criterion which we adopted in this category. In a related note, [CWA+96] demand support for sufficient data structures and algorithms.

Additionally, [KGS12] have suggested support for the object-oriented concept as an evaluation criterion. However, we think that this criterion is too implementation-centric (or at least design-centric) for specifications, with which the use of formal methods will usually start. Therefore we do not include this criterion here, though others might want to include it.

3.3 Supported development phases

[CWA+96] state that it should be possible to amortize the cost of a formal method or tool over many uses. For example, it should be possible to derive benefits from a single specification at several points in a program's life cycle: in design analysis, code optimization, test case generation, and regression testing.

Support for different phases of software development by formal methods varies widely. Many methods are designed for modeling, and in particular for *specification* including *validation* and *verification* of properties.

We think that a comprehensive, unambiguous and surveyable model is vital for any verification attempt (maybe with the exception of the verification of a few and small core algorithms only) because without such a model, it may not be clear what exactly is to be verified against what requirements.

A special phase which is not regularly present is that of *reverse engineering* – extracting the high-level functionality and a respective specification from a (typically ill-documented) legacy system. We owe attention to this additional project phase to [LYZ97].

Apart from the classical formal method subjects such as specification, validation, and verification, there is an explicit desire for support for the *architecture and design* phase by some authors, including [KDGN97], [WLBF09], [McG97], [CWA+96], [LYZ97], [KGS12]. There are also frequent wishes for *code generation* from formal models (e.g., [Sif97], [ACJ+96], [McG97], [LYZ97], [KGS12], [Bar98]) and support

for testing (e.g., [Sif97], [ACJ+96], [KDGN97], [WLBF09], [McG97], [CWA+96], [BH06], [LYZ97], [KGS12], [Don98], [Bar98]).

Bug diagnosis is an issue which deserves special mention beside verification, because finding that some property does not hold does not mean that one can then easily identify the source of error. Many proving tools provide traces which can be used to identify the problem, but the output is not always easy to use. [New14] points out the importance of this issue; [Bar98] even states that "industry is mostly interested in tools that find bugs rather than tools that prove correctness;" [CWA+96] explicitly mention "counterexamples as a means of debugging."

The potential benefits of formal methods in *maintenance* (as well as re-use) is highlighted by [ACJ+96], [KDGN97], [Bar98].

3.4 Technical criteria

In the category of technical criteria, we focus on tool support and how the method and the available tools interact with other aspects of system development from a technical point of view. This includes interfacing and interaction with requirements engineering and change management as well as with implementation.

The criterion of *overall tool support* is supposed to consider the variety of tools available for a particular method, and the general quality of those tools. Examples include editors, pretty printers, verification tools like (semi-) automatic provers and model checkers, interpreters for simulation, code generators, and test case generation. The need for tool support is stressed by virtually every relevant source. However, not all kinds of tools will be needed in every project, so the sheer number of available, different tools would not be an appropriate criterion.

Many tools for formal methods are free and even open source. This may be nice for a researcher, but IT managers in the industry may worry about long-term professional support. Thus [Sif97] brings up the criterion of *commercial support* for tools; [Bar98] argues in a similar direction.

Customizability of tools is also a highly relevant concern. This can be obtained by means of plug-ins, including alternative provers, checkers, animators, or editors, but also by various settings, e.g., to meet different general requirements for generated code.

An important issue stressed by many industrial sources regarding requirements engineering is the *traceability of requirements* throughout the development process. In case the product needs to be certified, this is a must. Consequently, also tools for formal methods should support tracking of requirements during specification, refinement, code generation, and test generation (at least). In the literature consulted specially for this work, the requirement of traceability is only found

in [Sif97] and [Don98] but we think that this should not lead to the conclusion that this issue was of minor importance.

Support for change management addresses the fact that the waterfall model is unrealistic in most settings, i.e., that we could go only once through each project phase. Requests for change as well as detection of higher-level errors frequently require us to revisit earlier project phases. The key questions are: How much stability of the initial specification is presupposed by the method? How easy it is to introduce a change in the specification if the main work has already shifted to design or implementation? How easy it is to validate and verify the amended model, and to incorporate the changes in the existing design and code? Note that this is not only important during development, but even more so for ongoing maintenance of the finished product. Interestingly, out of the literature consulted specially for this work, this criterion is only mentioned indirectly in [Don98] (who states that knowledge-based components can in practice only be incrementally specified) and, in a single place, in [LYZ97]. However, Börger and Stärk [BS03] stress this issue, and there are even several publications on the use of formal methods within agile development methods (e.g., [LÜ10], [PB15], [Wol12], amongst others). Our own experience shows the importance of paying respect to such a dynamic reality in industrial projects.

The *effect of the method on overall development time* is a crucial criterion for the industry. We list it as a technical criterion because both method and tool support play an important role in overall development. The effect of the use of formal methods on overall development time is certainly very difficult to measure or assess. We do not know of any empirically reliable, statistically significant comparisons, and thinking of real-life projects, it would be very hard and costly to make such investigations. So any evaluation in this respect is probably in the category of "educated guesses," considering different features of the method and its language and in combination with experience, trying to assess their likely or possible effects. A related, even more general criterion is given by [New14] as "high return on investment," which includes the demand that the method "quickly gives useful results," and also his wish that the method "improve time to market." [CWA+96] note under the keyword of "efficiency" that "turnaround time with an interactive tool should be comparable to that of normal compilation."

Regarding code generation, we can consider the efficiency of the generated code as well as the efficiency of code generation. The *efficiency of the generated code* is the quality of the code that has been generated by an automatic tool from a more abstract model: runtime behavior, use of memory, or the amount of manual fixing which is required after generation. This criterion is mentioned by [Sif97], [ACJ+96]. The

efficiency of code generation, on the other hand, concerns the speed (and use of resources) with which code is generated. This is important when the abstract model is subject to frequent changes or when modelers want to "play" with the model and test different designs, and thus they want to see the effects of their changes quickly. Also this criterion is given by [Sif97].

The demand for *interoperability with other methods and/or other tools* arises from the insight that different methods and tools are differently suitable for different tasks and project phases. Moreover, such a possibility will greatly enhance reuse, and will facilitate technology changes within a company. [KDGN97], [CWA+96], [Ban08], [Bar98] explicitly mention this criterion (with a more thorough discussion about the need to combine different methods by the latter). [WLBF09], amongst others, describe different projects in which at least two different methods were used to complement each other. [BH06] briefly discuss the combination of different formal methods (as well as hybrid methods).

A related criterion is *Integration of methodology and tools with the usual development methods and tools*, which is demanded by industry in order to facilitate the transition between different project phases, requirements tracking, tool-supported program verification against the specification, testing against the specification, unified storage of and access to all documents, and maybe most important of all, to keep things simple and familiar for developers. However, from the perspective taken in this article, it is not enough to integrate, e.g., satisfiability (SAT) solvers or model checkers into a programming environment, as we want to have also the earlier phases covered, including specification. This is why we did not include in our survey some papers specialized on such partial approaches. The issue of integration into common development environments is raised by [Sif97], [WLBF09], [CWA+96], [BH06], [Bar98].

3.5 Usability

In industrial settings, specially trained people will not be available for every development task, and will often not be available at all. The easier a method is applicable for normal engineers and developers, the easier it can be adopted by the industry. Moreover, certain products of the method should be accessible to people outside the development team, including domain experts (future users), managers, or even lawyers (considering that a formal specification should ideally be part of a contract between purchaser and supplier; cf. [KMGI14]). Thus arise criteria like general understandability, visualization and animation of a model.

The *learning curve* of a method concerns the speed with which an

average modeler (specifier, designer or developer) can learn the method from scratch and obtain useful results in practice. It includes the kind and amount of prior expertise needed, including a background in mathematics or familiarity with other formal methods. Respective criteria are nominated by [New14] ("easy to learn and easy to apply," but also "easy to remember"), [Sif97] ("time for learning," "ease of learning"), [ACJ+96] ("learning curve"), [CWA+96] ("early payback," "incremental gain for incremental effort," "ease of learning"), and [Bar98] ("Industry has no time to learn complicated new techniques"). (A related criterion is listed under *Industrial applicability* below, namely whether specially trained staff is required to use a method.)

General understandability is important because formal models often need to be understood not only by modelers themselves, but also, e.g., by domain experts in order to validate the model (*not only* experimentally via simulation but also thoroughly), managers who need to sign a contract based on a formal specification (amongst others), and maybe even lawyers who want to either defend or to contest whether the respective contract was fulfilled or not (cf. [KMGI14]). Understandability of requirements specifications is explicitly mentioned by [ACJ+96], [Don98], [Bar98]; related issues are raised by [LYZ97] (who praise graphical models for being "easy to comprehend") and [McG97] (who deals with appearance and syntax in this respect). Our own experience is that cryptic appearance of models constitutes a severe deterrence for representatives of the industry to adopt formal methods.

Documentation is an important issue as well, including reference handbooks as well as good tutorials. Reference handbooks are almost always available, but sometimes it is hard to get a good overview and it may be hard to find a particular construct, especially if one cannot remember or guess the exact name. Tutorials often present a few basic constructs but not more advanced constructs which are nevertheless often needed. It is also not rare to find that documentation is outdated, i.e., it has not been adapted to newer versions (a danger especially within small open-source communities with very limited resources). The issue of documentation is raised by some of the most industry-centric sources considered here, e.g., [New14], [Sif97], [BH95], [Bar98].

Support for collaboration is easily forgotten when academics develop a new method, but it is an important issue in larger real-life projects. Ideally, support for collaborative modeling should be the same as it is established for development. This is mentioned by [KDGN97].

3.6 Industrial applicability

The usability criteria dealt with above already have a lot to do with industrial applicability, as have many other criteria covered by all the

previous categories. However, there are still further criteria particularly concerning the capability of employing a formal method in a typical industrial setting. Industrial application very often means large and complex systems, as well as certain economic and legal constraints which do not, or only to a much smaller extent, apply to research projects. Also, in order to convince anyone in the industry to freshly adopt formal methods, it is necessary to win a priori confidence of the management, potentially on different levels of management from team leaders up to the very top. A number of related criteria will be detailed now.

The criterion of *support for industrial deployment* is designed to capture the availability of outside help. A relevant question in this context is, "Can I get reliable and, if required, long-time support from experts outside the company to introduce the method and to maintain its use?" A great number of industrial projects in which formal methods are employed rely on senior university students and teachers, but they may not be available beyond the scope of a few years – for instance, once a student has obtained their doctorate, they may move to some company, potentially a competitor, and even university teachers now usually have only short-term contracts (cf. [Bar98]). Consequently, the availability of commercial support (also beyond mere tool support) will be very helpful. However, also a good learning curve and good documentation (as listed above) as well as a stable community for the method and its tools are important factors. This point is mentioned by [ACJ+96] and [BH95].

Scalability is the ability to be well applicable to arbitrarily large and complex projects. One of the most common prejudices against formal methods is that they work only for academic "toy examples;" although this has long been shown to be untrue in general, one may still wonder how scalable a particular method really is, and some authors claim that there are considerable differences. Scalability as a criterion is explicitly mentioned by [New14], [Sif97]; [LYZ97], [Bar98] mention the state explosion problem as limiting the applicability of, e.g., Petri nets to large-scale systems, and explicitly note that, e.g., Z is "capable for large scale industrial applications."

Certainly the actual *amount of industrial experience* which has been gathered with formal methods in general as well as with particular methods so far is very interesting for decision makers who ponder newly introducing formal methods. Such information should also, if possible, detail what experience was gathered – what went well and what went wrong, which problems were encountered, how development time and costs were affected (so far as can be estimated), etc. Here [WLBF09] is a valuable source. [LYZ97] use a criterion which is probably largely based on such experience, namely, "industrial strength." Success rate is explicitly listed as a criterion by [Sif97]. Also [LYZ97] mention in a

few places whether particular methods have been successfully employed in the past.

A cliché that formal methods would require *specially trained, "expensive" personnel* is actually well-founded. It is one of the "Seven Myths of Formal Methods" of Anthony Hall [Hal90], who claims that the mathematics involved in writing, e.g., a Z specification would require only the basic "high school math" one should expect from any "practicing engineer"; however, we found that even people with a PhD in computer science are often put off by a Z or B specification, and with ordinary developers, the picture looks even worse. There are at least considerable differences among professionals in this respect; this can also, for instance, be seen with the widely differing opinions towards the understandability of Petri net-based specifications. Certainly, style of use matters a lot (cf. [KMGI14], for instance). Anyway, there are certainly considerable differences between particular methods in this respect, and one should certainly take into account the background of the staff which is currently available in a particular project setting.

Standardization can be very helpful for the industry: it enhances the probability of long-term availability of commercial tools and facilitates training as well as exchangeability of results. [Sif97] uses the more general keyword of institutionalization to cover standardization, amongst other factors for this kind of stability.

Related is the *availability and licensing of the method and related tools*. Most of the widely used methods and their tools are open source, which is definitely nice for researchers and students. However, open-source software requires a large and stable community to maintain and further develop. Moreover, the availability of commercial support and training is essential for more widespread uptake in the industry. Interestingly, this criterion is not explicitly mentioned in the literature surveyed (but see also the related criterion specifically for tools under technical criteria above).

4 Assessment of Selected Formal Methods

4.1 Overview

We will now use the criteria listed above to compare a range of specific methods which we considered relevant. We clearly favor state-based modeling techniques because they can be potentially integrated in model-driven engineering (cf. [OMG14]). The advantage is that the models used for rigorous specification can be more easily reused in later project phases, especially if they support refinement to arbitrary levels of detail. Thereby the efforts put into specification do not appear "wasted," which can be an incentive for managers and developers

alike. Moreover, state-based models in general support execution for the purpose of validation, model checking as well as code generation.

4.2 Alloy

Alloy is a state-based method based on "relational algebra, first-order predicate logic, transitive closure, and objects" [Zav15]. It is closely related to *Z* (see below), but allows only first-order expressions to render models fully analyzable [Jac12c]. A typical model written in Alloy is a collection of constraints that describes a set of structures, for example: all the possible security configurations of a web application, or all the possible topologies of a switching network [Jac12a]. Constraints are described in terms of relations. The standard tool for Alloy, Alloy Analyzer [Jac06], is a constraint solver (SAT solver).

4.2.1 Modeling criteria

Composition is supported through the composition operation [FPA04]. Temporal composition of functions is also possible through `merge` and `override` operators. Alloy supports the *refinement* mechanism, see for example the development of the Mondex system with Alloy [Ram08]. However, according to [New14], Alloy is not very flexible regarding the adjustment of the level of abstraction "as it does not support recursive operators or functions." *Concurrent* systems can be modeled in Alloy, see, e.g., [BRP+14], but according to [New14], due to limited expressiveness, the method is not suitable for large complex systems. As Alloy is a relational language, naturally *nondeterminism* comes with it for free [MK02]. However, this is also a problem because, as also noticed by [Zav15], nondeterminism can only be implicitly modeled in Alloy. Alloy has been used in designing systems where *global system properties of correctness* such as safety, security and reliability have played an important role – see, e.g., [BRP+14], [CK13] and [KJ08]. Alloy has no direct *notion of time* [GBF01]. However, it is still possible to express timing properties in Alloy models such as demonstrated by [WJ12], [DR12] and [ASR14]. [MRR11] selected Alloy for its "expressive power," amongst others, but according to [New14], the *expression of rich concepts* is *not* easy in Alloy.

4.2.2 Supported development phases

Alloy is a *specification* language. It has been extensively used for the specification of systems such as an electronic purse system [Ram08], an access control system [GBL12], a train protection system [XHM+12] and a flash file system [KJ08]. Alloy has also been extensively used for system *verification* and *validation* such as in the domain of distributed

collaborative editors [RIBQ13], tally systems [CK13] and an automatic train protection system [XHM+12]. Although [New14] noted verification as one of the plus points of Alloy, according to [Zav15], there are certain problems with the verification of progress properties with Alloy. [Wan10] also reports that Alloy is not suitable for complicated analysis tasks due to an enumeration-based analysis process as compared to a technique that is based on deductive reasoning. Validation is supported by the Alloy Analyzer's capability of simulation. The ease of *bug diagnosis* is middling, according to [New14]. The Alloy Analyzer obviously depicts counterexamples as graphs. Alloy supports the *design and architecture* of systems, see, for example, [BRP+14] and [BBG+08]. However, according to [KG06], there are certain problems related to the provided support. For example, the provided support is not sufficient for large models and it is difficult to relate counterexamples back to the source specification to find what flaw in the design caused the counterexample to be generated in the first place. There is some work about *code generation* from Alloy models such as translation of alloy models into Java code [Fer08]. However, the topic of code generation is not well addressed within the Alloy community. There has been an extensive work on using Alloy for *testing* and *automatic test case generation* – see, for example, [AAB+13], [RdAFLP12] and [KYZ+11]. However, the aforementioned model finding limitation is also valid for test case generation. [RH12] presented an approach for *reverse engineering* for checking the correctness of Java equality by modeling Java in Alloy. However, there is no practical demonstration of such an approach on any industrial case study.

4.2.3 Technical criteria

The primary *tool support* for Alloy is Alloy Analyzer. In its core, it is a SAT solver which can depict counterexamples as graphs. It also supports simulation by executing a model's operations. Further tools, including a higher-order constraint solver, code verifiers for Java, and an eclipse plug-in for Alloy, are listed in [Jac12b]. The Alloy Analyzer is download-able for free, including the source code, from the homepage[1]. However, there is no commercial support for this tool. There are problems regarding *traceability* as according to [KG06], it is difficult to relate counterexamples back to the source specification to find what flaw in the design caused it.

4.2.4 Usability

The *learning curve* is assessed by [New14] as good "for problems of modest complexity;" from [Zav15], we infer a medium learning curve.

[1] http://alloy.mit.edu/alloy/download.html

[New14] presents a relatively bad picture regarding the *understandability* of Alloy models, e.g., due to "a significant amount of syntactic overloading" and due to the possibility to combine different notational styles. [MRR11], on the other hand, selected Alloy for its "readability," amongst others. Our own impression is that at least more simple models are middling understandable by non-experts. [New14] assesses the available *documentation* as good. There is one standard textbook, *Software Abstractions: Logic, Language, and Analysis* by Daniel Jackson [Jac12d], and there are links to reference material and tutorials at the Alloy homepage.

4.2.5 Industrial applicability

Apparently, no *commercial support* is available for the Alloy method. *Scalability* problems are documented by [DKK+14], and [MRR11] even state that "Alloy was not designed to scale." Scalability is also assessed to be bad by [New14]. [WRL14] discuss ways to tackle scalability and performance problems in the verification of Alloy models; amongst others, they suggest to alter the style of modeling, which may, however, affect other model properties like intuitiveness. The Alloy homepage features many links to papers and case studies using Alloy. The case studies appear to be largely scientific yet thematically hint at a high degree of industrial involvement, which is even explicitly mentioned in some cases. We assess that *specially trained staff* is required for using Alloy. Alloy is not *standardized.* The Analyzer tool is open source.

4.3 The Abstract State Machine (ASM) Method

ASM is a state-based method based on work by Yuri Gurevich (see e.g., [Gur95]) and further developed to an industrial-strength modeling method by Egon Börger and others (see, in particular, [BS03]). The state space can be modeled by arbitrary data structures over possibly infinite sets. The core language is remarkably simple. Models can be made at arbitrary levels of abstraction.

4.3.1 Modeling criteria

Composition and de-composition is judged as medium by [Ban08]; however, from a practical point of view, we consider it to be quite flexible, although we have also experienced limitations with large systems composed of partly asynchronous subsystems. *Refinement* is good, according to [Ban08] as well as judging by our own experience. The ASM method allows for n-to-m refinement, which provides maximum flexibility and also makes abstraction possible (e.g., for reverse engineering). Refinement may combine changes of signature (data refinement)

as well as changes of control (procedural refinement) [BS03, pp. 25, 110ff]. ASMs are well suited for modeling *parallel and concurrent systems* [FSTW15]. The open issues with the theory for parallel ASMs do not affect practical work according to our experience, and also concurrent systems can already be handled in practice. ASMs allow for modeling *non-determinism* by an explicit "choose" operator; that is, a random element can be selected from a finite set. Moreover, rules and derived functions (or "macros") can be left abstract at any level of abstraction, leaving the outcome of their operations undetermined at this level. Of further help in this respect is the inclusion of an `undef` constant in any universe (or type, loosely speaking). *Global properties* can in principle be expressed via the state space. However, there is no explicit support for expressing, e.g., safety and liveness properties. There is also no explicit *notion of time* available. Regarding *special concepts*, there is work of Richard Banach and others regarding the modeling of continuous systems [BZSW14]. Regarding the *easy expression of rich concepts*, one advantage of the ASM method is its simple notation which can be easily adapted and expanded to meet the needs of a particular domain and project setting. The disadvantage of this is that tools will always only have a limited repertoire of such extra constructs, like standard set theory. Extra functions or predicates will have to be defined, e.g., as macros.

4.3.2 Supported development phases

Specification is arguably the chief purpose of the ASM method, where it has the advantage of a high level of general understandability (provided it is used accordingly) and easy integration with additional, natural text. *Validation* is supported first by its general understandability which allows walkthroughs with domain experts, but also by the availability of simulation tools. *Verification* is often done by hand when using ASMs. Thereby the level of proof granularity can be freely chosen, which can speed up proving considerably, under certain circumstances even as compared with automated proving. The Asmeta platform [GRS08] also enables model checking by tools. Also proof tools including PVS [ORS92], Isabelle [Pau94], HOL [GM93] and KIV [Rei92] have been used in the past to verify ASM models, but the required effort for this is very high. For *bug diagnosis*, the Asmeta platform provides the AsmetaMA model advisor. Regarding *architecture and design*, the refinement mechanism of the ASM method is useful, but limitations of decomposition also limit the usefulness of the method for this purpose. No off-the-shelf *code generator* is available for ASM models, but the refinement mechanism can be and has been used right down to programming code (manually). A *test generator* is available in the Asmeta platform. For proper *maintenance* of

a software system that has been developed using ASMs, supplements, fixes and other changes must be performed in the ASM model at the appropriate level of abstraction and from there on, the necessary refinement steps towards the code have to be repeated (manually). If required or desired, also the respective proofs have to be re-done. As no tool support is available for refinement and coding, it is questionable whether all this will really be done in practice; consequently, we assess the support of the ASM method for maintenance to be "poor." *Reverse engineering* has been successfully performed using ASMs (see e.g., [BS03, pp. 103ff, 349f, 362, etc.]).

4.3.3 Technical criteria

Overall tool support is at least medium. The tool CoreASM [FGG07], for which an eclipse plug-in is available, is popular for simulation. The Asmeta platform provides several tools, including simulating and testing tools, a test generator, a model checker, a tool for generating executable ASMs from Use Case models, and a special tool for service oriented components. The Asmeta platform is highly *customizable* as it can be easily expanded by additional plug-ins. Microsoft once integrated the ASM-based AsmL specification language in their development environments where it was used by their testing tool SpecExplorer [CGN+05], but support for this has long been discontinued. An open-source spin-off, XASM [Anl00], does not seem to be further developed or maintained either. *Commercial support* for any of these tools is not available from the developers. The effect of the use of ASMs on the *duration of a development project* is easily scalable. One can start using ASMs for the specification only, and then once people are familiar with it, make further use of it for design and coding in following projects. Verification is not a necessary part of the method and can be performed to any extent and also with any rigor deemed appropriate. Consequently, the ASM method can be freshly introduced in a team without having to fear tangible delays thereby caused, while in the long run, we expect the method to lead to an overall reduction in development time as debugging efforts will be considerably reduced and the testing phase will become shorter. (Note, however, that this can hardly be measured in practice.) In any case, extra effort is certainly lower than with most other formal methods. *Traceability of requirements* can be well achieved via the refinement mechanism, according to our experience; this is also supported by [Ban08]. In theory, *interoperability* with other methods as well as *integration in classical IDEs* may be possible as both CoreASM and Asmeta are based on Eclipse. Microsoft's AsmL could indeed be integrated in Visual Studio for some time, but this has not been followed up. At present, no such interoperability or integration is given.

4.3.4 Usability

The *learning curve* of the ASM method is short. There are only a handful of largely intuitive basic constructs to start with. After a short tutorial, developers should be able to use the method at least for more simple problems. ASM models can be made fairly *general understandable*. Although mathematical symbols can be used, the language is by design text-based. *Documentation* for the method itself is good – see, in particular, [BS03]. Documentation for the available tools is less good, however, consisting largely of scientific publications and manuals which lack overview and may even seem incomplete and/or outdated.

4.3.5 Industrial applicability

Professional *support for industrial deployment* is left to consulting by and co-operation with academic personnel but is not institutionalized, at least not on a perceptible scale. (However, e.g., our own institution, the Software Competence Center Hagenberg (SCCH) in Austria[2], provides such support.) *Scalability* has been proven by large-scale projects (see e.g., [SSB01], [BS03, esp. Chapter 9]). However, according to our own experience, it is somewhat hampered by practical issues with decomposition. There is some amount of *industrial experience*, also with large-scale projects – see e.g., [SSB01]; however, employment of ASMs seems to be scattered and not very widespread. We do not think that *special staff* is required for modeling with ASMs, though verification certainly will require better trained staff (such as computer scientists or mathematicians). There does not currently exist any *international standard* for ASMs. There is no *license* required for applying the method, and all the tools mentioned are open source.

4.4 B

B is a formal language for modeling software specifications and reasoning about them. It is based on set theory and standard first-order predicate logic. B is supported by a commercial tool platform, Atelier B [Lec14].

4.4.1 Modeling criteria

Composition and de-composition is supported by the possibility to call operations of other machines and to access, e.g., data structures from other machines. This works basically like calling procedures in procedural programming languages, but B additionally provides a few options regarding the visibility and accessibility of elements of other ma-

[2] http://www.scch.at

chines. Every machine has its own file. According to Banach [Ban08], "The [...] INCLUDES, USES, SEES mechanisms are certainly composition mechanisms, but they just act at the top level." B supports only a one-to-one notion of *refinement* (cf. [Ban08]). In practice, refinement relies very much on defining the actions of operations which can initially be left with an empty action, "`skip`." That is, in the operations of one machine you can call operations of other machines which may initially be left abstract. B does support *parallelism*, except for code generation, but it does not support *concurrency* [KGS12, LYZ97]. B supports *nondeterminism* by allowing for nondeterministic choice of values for variables out of a given set (corresponding to Hilbert's ϵ-operator) as well as by operators "`ANY`" (unbounded choice of value) and "`CHOICE`" (nondeterministic choice of alternative substitutions). Regarding *system properties*, it is possible to express typical safety properties through invariants in B, but there is no way to elegantly express, e.g., temporal properties, as B has no explicit means for the modeling of *timing* or *temporal properties* (see also [LYZ97]). An extension of the method has been proposed by [AM98] in this direction. *Reliability* properties can be expressed via invariants, and some reliability properties can moreover be checked via the model checker ProB [LB08]. Regarding the *easy expression of rich concepts*, B provides a rich language for set theory and, in particular, relation (and function) theory. However, expressing certain concepts such as data structures in such a language can often be awkward and unintuitive.

4.4.2 Supported development phases

According to our own experience (but also according to [WLBF09]), B is well suited for formal software *specification*. For *validation*, animation of a B specification can be performed with the tool ProB. The commercial tool-set Atelier B provides a proof obligation generator and an interactive proving environment with different provers for *verification*. Atelier B uses an axiomatic proof system (cf. [LYZ97]). ProB adds the possibility of model checking. Regarding *bug diagnosis*, the ProB model checker provides a counter-example with a respective trace. According to [LB08], "[...] if ProB finds a counterexample, the user gets important feedback: the proof obligation cannot be discharged, along with a reason why." According to [KGS12], B can also be used for *design*. However, according to our own acquaintance with the method, at least for larger pieces of software, we strongly recommend to use other design tools alongside as well such as graphical modeling support. Atelier B comes with *code generators* for different target languages, including C, C++, Java, and Ada. Although the generated code requires some post-processing, it is a good basis for the implementation of a B specification. Support for generating test cases

is available for B as mentioned by [KGS12]. [LYZ97] also mention the role of the animator ProB in this direction. B has been successfully used for *reverse engineering* in the railway domain[3]; however, in our own experience, the one-to-one notion of refinement with unidirectional simulation does not make B very suitable for abstraction as required for reverse engineering.

4.4.3 Technical criteria

Tool support for modeling and analyzing in B is available in the form of the Atelier B platform. Atelier B has been professionally developed and is available with a *commercial* license and, alternatively, a (somewhat restricted) free license. The commercial license includes professional support. Atelier B includes an editor, syntax and type checkers, a proof obligation generator, automatic provers and an interactive proving environment, as well as code generators for different target languages. A stand-alone model checker, ProB, is available for B and can be used together with Atelier B. *Customization* of the tool is restricted. Support for *requirements traceability* in the B method is discussed by [PD06] and [dSRJ10]. The use of the B method may increase *development time* initially as compared with no use of formal methods. The amount of time spent in the specification and verification phases will depend on how many proof obligations can be discharged automatically and how complex the remaining proofs are. Experience shows that proof obligations can soon become rather intricate. However, once the specification is proven, less effort for debugging and testing may overcompensate the effort. This may still result in an overall reduction of development time. The *quality of generated code* is fair. However, post-processing of the generated code is required and there are restrictions regarding possible data types. The code generation process works reasonably fast. *Traceability* can be achieved to the extent that requirements can be associated with different machines. *Interoperability* is possible with Event-B (the tool Atelier B supports both methods).

4.4.4 Usability

We assess the *learning curve* of B to be relatively high. The language is based on predicate logic and set theory which may be familiar to many, though not all stakeholders. However, there are many symbols and constructs which are not familiar to most non-mathematicians, and certain relational constructs require a kind of thinking to which mathematicians are accustomed but not necessarily developers or designers. For a modeler, it is also necessary to get into proving from the

[3] http://www.data-validation.fr/data-validation-reverse-engineering/

very start, enforced by both the method and the tool. Likewise, *general understandability* is medium at best. Many domain experts will struggle to read a B specification without initial training, and lawyers, for instance, can be expected to find it very difficult. *Documentation* is good. There is the B-book [Abr96], and Atelier B comes with extensive documentation for all its features. The work presented in [BCF+97] describes several related case studies.

4.4.5 Industrial applicability

Professional support for *industrial deployment* of the B method is provided by several companies such as ClearSy[4], Systerel[5] and SCCH. *Scalability* is given via decomposition into different machines, which happens almost automatically during refinement and which prevents large, unwieldy artifacts. However, the increasing number of machines (and thereby source files) can easily lead to another kind of loss of oversight. Moreover, a large number of interdependent machines also leads to intricate proof obligations, many of which can not be automatically discharged any more. The same holds if complex data structures (e.g., large records) are used. On the other hand, it must be noted that B has been successfully used in large-scale industrial projects; thus we rate scalability as good, with some caution. B has been used in major *industrial projects* since the late 1980s; [WB95] and [Bou14] mention several such projects. [LYZ97] attest that B enjoys "great industrial strength." Regarding the *requirement of special staff*, our comments on the learning curve above also suggest that modeling will usually have to be performed by computer scientists or mathematicians, or maybe the odd developer with a special interest in algebra. There is no *international standard* covering B. The basics of the method are described in the *B-book* [Abr96] and in a freely available reference manual [Cle15]. For the tool Atelier B, there are a *commercial license* (with support) as well as a free license (with some restrictions, especially without code generators other than for C) available.

4.5 Event-B

Event-B [Abr10] is a formal language for modeling and reasoning about large reactive and distributed systems. Event-B has been derived from the classical B method and is therefore also based on set theory and standard first-order predicate logic. Event-B is provided with Rodin [ABH+10], a platform which supports the writing and proving of specifications.

[4] http://www.clearsy.com
[5] http://www.systerel.fr

4.5.1 Modeling criteria

Regarding *composition and de-composition*, [Ban08] assesses Event-B as "good." However, while applying it to develop a real-life safety-critical system [MBDT15, Mas15], we found out that the model decomposition/recomposition facilities in Event-B are not straightforward and require further improvement. Event-B supports a rather restricted notion of *refinement* where each machine is further refined by only one machine. Several techniques have been proposed to liberalize this linear refinement process, e.g., observation-level-driven formal modeling [MBDT15]. Event-B does not explicitly support *parallelism and concurrency* (note that a paper by Abrial introducing events explicitly speaks of the development of *sequential* programs [Abr03]). However, both parallel (cf. [HA10]) and concurrent (cf. [BDSW14]) programs can be defined using the related notions of decomposition and refinement. Event-B does support *nondeterminism* by allowing for nondeterministic choice of values for variables (Hilbert's ϵ-operator) and by allowing for event parameters. *Global system properties* can effectively be specified in Event-B using invariants. Event-B has no explicit means for the *modeling of timing or temporal properties*. However, there are several proposals, e.g., [AM98] and [RC07], to express such properties in Event-B specifications. Regarding *special concepts*, there exist proposals regarding hybrid and continuous systems; see, in particular, [BZSH11]. Regarding the *easy expression of rich concepts*, Event-B provides a rich language for set theory and, in particular, relation (and function) theory. However, expressing certain concepts in such a language can often be awkward and unintuitive.

4.5.2 Supported development phases

Event-B is certainly well suited for formal *specification*. [Sin13] lists several examples of how the Event-B method has been used for the specification of critical systems. Animation is the most commonly used technique for *validation* in Event-B. The animation plug-in ProB is already available for the Rodin platform. Event-B provides a variety of tools regarding *verification* such as Atelier B provers [MMFA12], Isabelle/HOL for Rodin [Sch11], SMT solvers for Rodin [DFGV14] and the model checker ProB. Regarding *bug diagnosis*, the ProB model checker provides a counter-example with a respective trace. A couple of *code generators* have been developed for Event-B. However, of the four tools we found for generating C code, one is not publicly available [FHB+14], one was custom-built and only covers a part of Event-B syntax [Wri09], EB2ALL explicitly requires manual post-processing [MS11], and Tasking Event-B [EBM+12] came out of an academic project which is discontinued; consequently, Tasking Event-

B only works with old versions of Rodin and requires specific versions of several other plug-ins. All in all, code generation for Event-B does not appear to be mature and well supported. The MBT (Model Based Testing) plug-in [DIMS12], which is is available for the Rodin platform, is capable of generating *test cases* from a formal specification. However, its effectiveness for a real application is questionable. We do not know of any special support for *maintenance* by some tool for Event-B. Certainly, if code generation works, than additional features, changed features or bug fixes can be affected on most abstract models by means of (generalized) refinement and then the respective further refinement steps must be performed (with possible re-use of proofs) until re-generation of code becomes possible; but this is possible with other methods as well. We do not have information about the use of Event-B in *reverse engineering*; however, the one-to-one refinement concept is certainly a great obstacle for that task.

4.5.3 Technical criteria

Overall tool support for Event-B is good, and some of the tools are already well-proven in use with B. The main tool is the Eclipse-based platform Rodin, a highly *customizable* platform into which extra tools can be plugged, including alternative editors (e.g., Camille [BFJ+11]), different provers and model checkers (e.g., ProB), a requirements traceability tool (ProR [HJL14]), animation and visualization tools, documentation tools, etc. Rodin itself provides the look-and-feel familiar to all developers who use Eclipse. An alternative tool is Atelier B, which supports both the classical B method and Event-B. *Commercial support* for Event-B is provided in the form of commercially supported tools such as Atelier B and ProR. *Requirements traceability* is supported by a requirements editing and tracing tool for Event-B, ProR. Initially, the use of Event-B may increase *development time* as compared with no use of formal methods. The amount of time spent in the specification and verification phase will depend on how many proof obligations can be discharged automatically and how complex the remaining proofs are. Experience shows that proof obligations are in general less complex than those for comparable B models, so that more of them can be automatically discharged. Once the specification is proven for correctness, less effort for debugging and testing compensates the additional effort. *Interoperability* is possible with *B*, especially when using Atelier B. [MMS08] presents an approach for model checking Event-B specifications by their encoding into Alloy. [RC14] present an approach for translating Event-B to JML. [MFGL10] present an approach for translating Algebraic State Transition Diagrams (ASTD) [FGL+08] into Event-B. [SB06] provide a UML like graphical front end for Event-B. *Integration in development environments* may be rel-

atively easily possible as Rodin is a plug-in for the widely used Eclipse IDE, but no concrete efforts in that direction are known to us.

4.5.4 Usability

We assess the *learning curve* of Event-B to be medium (very similar to that of classical B). The language is based on predicate logic and set theory which may be familiar to many, though not all stakeholders. However, there are many symbols and constructs which are not familiar to most non-mathematicians, and certain relational constructs require a kind of thinking to which mathematicians are accustomed but not necessarily developers or designers. For a modeler, it is also necessary to get into proving from the very start as the method, and also the tool Rodin, force to do it. Rodin discharges most of the proofs automatically, but some may require interactive discharging. Likewise, *general understandability* is medium at best. Many domain experts will struggle to read an Event-B specification without initial training, and lawyers, for instance, can be expected to find it very difficult. *Documentation* is good. There is the Event-B book [Abr10], but on the Event-B and Rodin homepage[6] one can also find a good handbook and tutorial for Rodin, language reference, further examples, and a wiki with all kinds of information. Some *support for collaboration* is given via a plug-in called "Team-working feature" (see entry "Team-based development" at Event-B/Rodin homepage), which enables the use of SVN or a similar version-control tool.

4.5.5 Industrial applicability

Professional support for *industrial deployment* of the Event-B method is provided by various research and development establishments such as ClearSy as well as the SCCH. *Scalability* is given via refinement, decomposition, patterns and generic instantiation. While refinement is already a well-developed notion in Event-B, the other techniques suffer from certain limitations; see for example [Mas15]. In comparison with classical B, proof obligations tend to stay simple even as the model grows. Event-B is a popular method in industry. However, as noted by [New14], most industrial projects modeled using Event-B are about control systems. Some of the examples of application of Event-B to industrial problems are transportation systems [SBRG12], medical systems [Mas15], and business information systems [BFLM05]. Regarding the *requirement of special staff*, our comments on the learning curve above also suggest that modeling will usually have to be performed by computer scientists or mathematicians, or maybe the odd developer

[6] http://www.event-b.org/

with a special interest in algebra. There is no *international standard* covering Event-B. It is available under an open-source license, as are most of the tools.

4.6 TLA+

TLA+ [Lam02] is a further development of the state-based specification language Temporal Logic of Actions (TLA) [Lam94]. It has been designed by Leslie Lamport at Compaq for the specification of concurrent systems in particular.

4.6.1 Modeling criteria

TLA+ supports different mechanisms for *composition* (see [Lam02, Chapter 10]) such as through stuttering [Mer08]. *Refinement* is assessed as "good" by [New14, p. 28]; and according to [Mer08, p. 445], "A distinctive feature of TLA is its attention to refinement and composition". Support for modeling *parallel, concurrent, and distributed systems* is good, as can be expected from the original purpose of the language; this is confirmed by [New14, p. 36]]. As for *nondeterminism*, Lamport describes the `CHOOSE` operator as deterministic, but then goes on to give an example of a non-deterministic specification [Lam02, Section 6.6]. TLA+ does not formally distinguish between specifications and *system properties*: both are written as logical formulas, and concepts such as refinement and composition [Mer08]. It uses set theocratic constructs to define safety properties and temporal logic to define liveness properties. It also provides supporting tools to verify these properties such as the TLA+ Proof System (TLAPS)[7] and the TLC model checker [YML99]. *Modeling of time* is explicitly supported by TLA+ [NRZ+15, p. 69] (cf. [New14, p. 27]), which enables the modeling and checking of *performance properties*. According to [New14, pp. 27, 36], *rich concepts can be easily expressed* in TLA+.

4.6.2 Supported development phases

TLA+ has been primarily designed for *specification*. Simulation for the purpose of *validation* is possible through the TLC model checker. *Verification* is supported through the TLC model checker and the TLAPS interactive proof system [CDLM10]. According to [New14, p. 31], TLAPS is good, while the power of the model checker is "limited" [New14, pp. 28, 35]. *Bug diagnosis* is middling, according to [New14, p. 29]. *Performance checking* is possible (see under *modeling of time* above). TLA+ models provide confidence that the models faithfully

[7] http://tla.msr-inria.inria.fr/tlaps/content/Home.html

reflect the intended system and serve as a basis for more detailed designs and ultimately for implementations. Therefore, TLA+ implicitly supports the *architecture and design* phase. The PlusCal compiler generates a TLA+ specification that can be verified using TLC. It has a C style syntax that can be used for generating executable code. However, no automatic *code generation* utility is available. The ProB tool can be used for animating and model checking TLA+ specification by translating TLA+ to B [HL12]. It can also be used for automated *test case generation*.

4.6.3 Technical criteria

There exists a few tools for TLA+ but the *overall tool support* is limited. The TLA+ toolbox, available from the TLA+ homepage[8], is an IDE which is available for free. It comprises an editor, a pretty printer, a model checker, and an interactive proof tool. Also the animator and model checker ProB, originally developed for B and later adapted for Event-B, meanwhile supports TLA+. [New14] assesses tool support to be good. According to [New14] and [NRZ+15], extra *time effort* for using TLA+ is less as compared to other formal methods. As a TLA+ specification can be translated to B and vice versa, tools for one method can supposedly also be used for the other.

4.6.4 Usability

The *learning curve* is described by [New14, p. 36] to be very short. According to [New14], TLA+ specifications are reasonably well *understandable*. The alternative language PlusCal, which can be automatically translated to TLA+, appears to be even better understandable for developers due to its C-style syntax. Regarding *documentation*, there is a good introductory book by Leslie Lamport which is available for free [Lam02]. The TLA+ homepage lists some other useful resources as well.

4.6.5 Industrial applicability

TLA+ is supported by Microsoft Corporation, but we have no indication for *professional deployment support*. However, according to [New14], such support may be less needed for TLA+ than for other methods due to the good learning curve. TLA+ has been used for several large projects. Some examples of application of TLA+ to *industrial applications* are discussed in [BL03]. [New14, p. 27] states that TLA+ has been used successfully on many projects in industry. To name a few, TLA+ has been successfully used by Compag [BL03],

[8] https://research.microsoft.com/en-us/um/people/lamport/tla/tla.html

Intel [BL03] and Amazon [NRZ+15]. According to [New14], no *special staff* is required to use TLA+ – normal developers can use it easily. Neither TLA nor TLA+ have been *standardized* by an international organization. TLA+ and its standard tools are open source.

4.7 VDM

VDM – the Vienna Development Method – is one of the oldest formal methods; it was developed in the nineteen-seventies at Vienna offices of IBM by a group around Heinz Zemanek and Dines Björner. It has three dialects: 1) VDM-SL [Bic98], which allows for the specification of abstract data types with pre- and post-conditions of data-type-related functions as well as for state-based modeling, 2) VDM++ [FLM+05], which extends VDM-SL with features for object-oriented modeling and concurrency, and 3) VICE (VDM++ In Constrained Environments), which extends VDM++ with features for describing real-time computations [MBD+00] and distributed systems [VLH06].

4.7.1 Modeling criteria

Composition is possible according to [LYZ97], but [KGS12] and [McG97] deny it. Classical VDM models can be structured into data types (and those into functions) and modules, while the object-oriented dialect VDM++ can be structured into classes, thus we see composition techniques as given. *Refinement* is achieved through data reification and operation decomposition. While the former transforms abstract data types into concrete data structures, the latter transforms specifications into a form that is implementable in a programming language. Support for the notion of refinement in VDM can be rated as good. According to [LYZ97] and [McG97], VDM does not support any sort of *parallelism*. However, a much more recent technical report [LW13] describes the use of VDM for *distributed* embedded systems, and explicitly deals with the challenges of *concurrency* and *real-time systems*. Moreover, [PB13] state that "VDM emphasizes on the feature of concurrency control;" we rate these much newer sources to be more reliable in this respect and take the support of concurrency to be given. VDM has also been used for the modeling and validation of distributed embedded systems [VLH06]. Support for *nondeterminism* is only given in the object-oriented dialect VDM++ [KGS12]. [McG97] states that VDM has no explicit *notion of time*. However, a much more recent technical report [LW13] describes timing analysis for identifying performance bottlenecks using the VDM tool Overture[9]. The work presented in [VLH06] and [MBD+00] also deals with real-time systems. Regarding

[9] http://overturetool.org/

special modeling concepts, [McG97] and [PB13] note that VDM does have explicit *exception handling*. However, [LYZ97] note that there are no communication concepts, and [McG97] notes that there is no support for performance, reliability, usability, or user interface modeling. Support for some of these concepts such as performance and reliability has been introduced to the method since then.

4.7.2 Supported development phases

VDM is first and foremost a *specification* method. Apart from specification, the supporting commercial and open-source tools, such as VDMTools [FLS08] and Overture, allow *validation* and *verification* through proofs and debugging. VDM has been used in the *design* phase of projects [McG97], [WLBF09]. [PB13] note, however, that VDM does not support all aspects of design. *Code generators* are available both within the commercial VDMTools (from VDM++ to C++ or Java) and within the open-source Overture (for Java). Both VDMTools and Overture have tools to analyze *test coverage*, and the latter also includes a tool for combinatorial testing. [LYZ97] state that VDM is unsuitable for *reverse engineering*.

4.7.3 Technical criteria

There are both commercial and open-source *tools* available for VDM. VDMTools has been developed *commercially* but is available for free now. It includes, amongst others, an interpreter and debugger, a test coverage statistics tool, and code generators for C++ and Java (for VDM++ models only). The Overture tool is an open-source, Eclipse-based IDE. It includes, amongst others, tools for interpretation and debugging, for combinatorial testing and analyzing test coverage, managing proof obligations, and code generation for Java; as it is built on Eclipse, it is well *customizable* as it will be easy to add further plug-ins. However, [PB13] say VDMTools lack usability, noting that "there is no internal editor for models," so one has to use external editors. They also say that "the error list cannot be emptied and so it is hard to see which errors are new and which have already been fixed." A recent work presented in [CLH+15] details how the Overture platform can be extended to *support other methods and tools* such as theorem proving through Isabelle, model checking through Microsoft Formula[10], simulation through ProB and refinement through Maude [CDE+99].

[10] http://research.microsoft.com/en-us/projects/formula/

4.7.4 Usability

We expect the *learning curve* for developers to be rather fast, as the language is, in some ways, similar to programming languages. *Understandability* for developers is therefore also rather good, but not for people outside software development (even though [PB13] assess VDM specifications to be "easy to understand"). Sufficient *documentation* is available for both the method and tools including manuals and tutorials.

4.7.5 Industrial applicability

VDM is supported by a commercially available set of tools bundled as VDMTools which is developed and maintained by CSK Systems. Manuals, tutorials and license for the tools are also available freely. Industrial projects in which VDM was used, with resulting code of up to 197 KLOC [McG97], suggest that VDM is *scalable* to a considerable degree. There is a considerable amount of *industrial experience* with VDM, see for example [McG97, WLBF09]. Our impression from example models is that software engineers should be able to learn how to model in VDM fairly quickly. However, formal verification will certainly require *special staff*. VDM is standardized by the International Standardization Organization as ISO/IEC 13817-1:1996 [ISO96]. The method itself is thereby *licensed* by the ISO.

4.8 Z

Z is a state-based specification method with a language based on Zermelo-Fraenkel set theory (hence the name) and predicate logic. It was developed by the Programming Research Group at Oxford University under Jean-Raymond Abrial around 1980 [ASM80].

4.8.1 Modeling criteria

Composition (and de-composition) can be achieved by means of so-called "schemas" [Ban08], [Bow01] or by means of "promotion" [Ban08]. [Ban08] notes that the schema calculus is not monotonic with respect to refinement. From comments in [Ban08] and [KGS12], we reckon that the quality of the composition mechanisms of Z should be assessed as medium. *Refinement* is possible [Ban08], [Bow01], [LYZ97]; however, [Ban08] notes, amongst others, that "spurious traces, not corresponding to real world behaviour, can be generated." Refinement cannot completely go down to the code level, as [Bow01] writes: "Some data and operation refinement is possible in Z but at some point a jump to code must be made, typically informally." All in all, the refinement mechanisms of Z appear to be of middling quality at best. Z does

not directly support the modeling of *concurrency* [LYZ97], [McG97], [PB13]. However, the work presented in [SD01] proposed the integration of Object-Z, an object-oriented extension of the Z method, and Communicating Sequential Processes (CSP) for specification, refinement and verification of concurrent systems. [BL96] also proposed a framework for the specification of *parallel* and *distributed* real time systems. Regarding support for *nondeterminism*, we found contradicting statements: according to [Bow01] and [FHP06], this is given, but according to [KGS12], it is not possible. Actually, Z does not support nondeterminism explicitly. However, for instance, the possibility of several after-state valuations for a single pre-state binding is a clear notion of nondeterminism in Z. This idea is further explained in [MHM02]. *Expression of global system properties* such as safety and liveness is not straightforward in Z. The work presented in [Hau91] makes this clear but also suggests a way how such properties can be dealt with in Z specifications. An explicit *concept of time* is obviously not given for the standard Z method [LYZ97], [McG97]. Regarding *special concepts*, [McG97] and [PB13] note that there is no explicit exception handling and no support for user interface modeling. The language of Z is certainly rich, but whether it allows for *easy expressions* appears to be questionable for us.

4.8.2 Supported development phases

Z is a *specification* language in the first place. It supports *validation* via simulation [Bow01] and *verification* via theorem proving as well as model checking [Bow01]. According to [LYZ97], there is a "semi-automatic proof system" available for Z. The possible use of Z in *design* is mentioned in [KGS12], [LYZ97], [PB13], and the latter two even call Z "strong" in this respect, though [PB13] qualify their assessment, saying that Z was not supporting all aspects of design. [KGS12] state that no code generator is available for Z, and [PB13] even say that a "specification written using Z notation cannot be used to generate computer source code directly;" however, the 1997 paper of McGibbon [McG97] says that code generation is supported. We ourselves failed to find a code generator, so we conclude that *code generation* is probably not available. [KGS12] write that *test generation* for Z is "strong," and [PB13] write that Z "provides a strong base" for *testing*. Usefulness of Z in *reverse engineering* is good according to [LYZ97].

4.8.3 Technical criteria

There is at least medium *tool support* for Z. Tool sets available include the Community Z Tools (CZT) [MU05], an open source framework with an Eclipse-based IDE for parsing, theorem proving, type-checking,

transforming, animating and printing ISO Standard Z conforming specifications. As CZT is built on Eclipse, it is well *customizable* as it will be easy to add further plug-ins. The animator and model checker ProB meanwhile also supports Z. There is a version of the open-source higher-order logic proving software ProofPower for Z [Art11]. Another proving tool for Z is z-vimes[11]. High-performance proving is also possible with HOL-Z, which is "a proof environment for Z built as plug-in of the generic theorem prover Isabelle/HOL" [Wol08]. [ORA09] provides some detail about the former *commercial tool support* available for Z. Currently, a British company, Lemma 1 Ltd.[12] provides commercial support for the Z method and the tool ProofPower. Regarding its effect on *development time*, our evaluation suggests that extra time is required especially as compared with methods like ASMs or TLA+ because of its "not-so-familiar" notation. [KDGN97] also presented a similar observation. Following [Ban08], we rate *traceability* in Z to be medium. Regarding *interoperability* with other methods, the ProB tool allows animation of specifications written in the Z language. HOL/Isabell can also be used to verify Z specifications.

4.8.4 Usability

We estimate the *learning curve* for Z to be long. Both [Bow01] and [KDGN97] confirm this observation. We rate the general *understandability* of a Z specification to be rather bad. [Bow01] (amongst others) notes that any Z specification should be accompanied by natural text to explain each schema; however, we see two problems here: First, the formal specification and the natural text cannot say the same thing, because the latter lacks precision; thus there is no guarantee that the reader of the natural text, e.g., a domain expert or a manager, will have the same understanding of the specified system as the developer who (hopefully) uses the formal part, and conflicts may occur later. And secondly, experience shows that it is not very likely that in case of later changes in the specification, the natural text will be updated as well according to the changed formal parts, so formal specification and explaining natural-language text are likely to drift apart. The problem of understandability is exacerbated by Z's frame problem [Ban08] which forces specifiers to explicitly state for every variable which maintains its value in a step that it does so, thus unnecessarily cluttering the specification. According to [Bow01], *documentation* for Z is good.

[11] http://sourceforge.net/projects/z-vimes/
[12] http://www.lemma-one.com/

4.8.5 Industrial applicability

A British company, Lemma 1 Ltd., provides *commercial support* for the Z method and the tool ProofPower. *Scalability* is rated as good by [LYZ97]. However, [New14] did not find application examples involving the verification of large systems. According to [WLBF09], there is much *industrial experience* with Z. Also [McG97] gives several industrial examples. We are certain that *special staff* is required when using the Z method. This is at least partly confirmed by [Bow01], who states that when using Z/EVES for proving, it "takes a significant amount of skill to use effectively." Z is *standardized* by an international body – as ISO/IEC 13568:2002 [ISO02]. This also settles the *licensing* for the method itself; the currently available tools are open source.

4.9 Tables for Comparison

We now try to allow for a direct comparison between different methods through simplified tables, see Tables 1 to 5.

"Y" means "yes/supported," but quality unknown; "N" means "not supported." A dash "-" means that we could not find (sufficient) information in the literature on this point. A "?" means that we have inconsistent or even contradicting information on this point. "(Y)" indicates restricted support, "(N)" indicates little support, "(Good)" means "Good" with some proviso, etc.; parentheses may also indicate that special versions or prototypes support this feature, but not the standard version. "Med." abbreviates medium (middling) quality, "Part." abbreviates partial support, "Man." abbreviates manual, i.e., no tool support, "Impl." abbreviates only implicit support, and "Adapt." abbreviates adaptable. "Cm." abbreviates "commercial" (licensing), "OS" open source. "n/a" means "not applicable."

In Table 2, *specification* is omitted as it is supported by every method considered, *validation* is also possible for every method considered through animation or a technique of similar quality, *performance checking* is omitted as we could not find any information about either of the methods in question. In Table 3, the criterion of *change management* is omitted due to lack of information.

4.10 A Simplified Table for Project-specific Assessment

Furthermore, we try to condense the available information into a much simplified table for fast management decisions – see Table 6. Comparison categories include the project type, the company, and the strategic goal pursued by the introduction of formal methods. This table is not only simplified, but also rather experimental and probably of de-

	Alloy	**ASMs**	**B**	**Event-B**	**TLA+**	**VDM**	**Z**
(De-)Compos.	Y	Med.	Med.	Med.	Y	Y	Med.
Refinement	Med.	Good	Med.	Med.	Good	Good	Med.
Parall./concur.	Med.	Good	Part.	N	Good	Y	N
Nondeterminism	Impl.	Y	Y	Y	Y	(VDM$_{++}$)	Y?
Global propert.	Y	Med.	Med.	Y	Y	N?	(N)
Time/perform.	(N)	N	N	N	Y	Y	N
Spec. concepts	-	(Hybr.)	N	(Hybr.)	-	Few	-
Rich conc. easy	(N)	Y	Med.	Med.	Y	(N)	?

Table 1: Modeling criteria

	Alloy	**ASMs**	**B**	**Event-B**	**TLA+**	**VDM**	**Z**
Verification	(Good)	Med.	V.Good	V.Good	Good	Y	Y
Bug diagnosis	Med.	Y	Y	Y	Med.	Y	-
Archit./design	Med.	Med.	(Y)	-	(Y)	(Y)	Good
Coding	Poor	Man.	Y	Poor	(N)	Y	N?
Testing	Med.	Y	Y	Poor	Y	Y	Good
Maintenance	-	Poor	-	-	N	-	-
Reverse engin.	(Y)	Y	(Y)	-	-	N	Good

Table 2: Supported development phases

	Alloy	**ASMs**	**B**	**Event-B**	**TLA+**	**VDM**	**Z**
Tool support	Y	Med.	Good	Good	(Good)	Med.	Y
Comm. support	N	N	Y	Part.	N	Y	Part.
Time effort	-	Adapt.	(Long)	(Long)	(Short)	-	(Long)
Efficient code	-	n/a	Med.	n/a	n/a	-	n/a
Efficient code gen.	-	n/a	Y	n/a	n/a	-	n/a
Traceability	Poor	Good	(Y)	Good	-	-	Med.
Interoperability	N	(N)	Part.	Part.	Part.	N	N
Integration/IDE	-	(N)	N	(N)	-	-	(N)

Table 3: Technical criteria

	Alloy	**ASMs**	**B**	**Event-B**	**TLA+**	**VDM**	**Z**
Learning curve	Med.	Good	Med.	Med.	Good	(Good)	Bad
Understandability	Med.	Good	Med.	Med.	Good	Bad?	Bad
Documentation	Good	(Good)	Good	Good	Good	Good	Good
Collaboration	-	N	-	Y	-	-	-

Table 4: Usability

	Alloy	**ASMs**	**B**	**Event-B**	**TLA+**	**VDM**	**Z**
Deployment sup.	N	(N)	Y	Y	N	Y	Y
Scalability	Bad	Med.	(Good)	Med.	-	Y	(Good)
Experience	(Much)	Med.	Much	(Much)	Much	Much	Much
Special staff	Y	(N)	Y	Y	N	(N)	Y
Standardization	N	N	N	N	N	Y	Y
Licensing	OS	OS	Cm.	OS	OS	Cm/OS	OS

Table 5: Industrial applicability

batable accuracy, even if it was derived from the previously presented assessments.

A few criteria may warrant explanation. In order to start employing a particular method "right away," a company will probably require outside help by trained and experienced people; the question arises whether initial help – for a short time only – suffices or whether the company will either have to hire specially trained staff eventually or continue to get outside help. On a related note, especially (though maybe not only) small companies will often not be able to economically afford a transition phase in which the introduction of a formal method will slow down work, never mind what benefits could be gained in the long run (see category "Company"). By "goal" in the last category, we mean the main objective of introducing formal methods.

In this table, a dash means that we could not gather enough evidence to pass a judgment regarding that particular question, and question marks signify particular uncertainty. Please note that other entries in this table are also conjectural.

5 Conclusion

The main aim of this work is to consolidate and further develop a system of criteria for assessing particular formal methods especially with respect to their potential usefulness in industrial projects. We hope that our work will contribute to better acceptance of formal meth-

		Alloy	**ASMs**	**B**	**Ev-B**	**TLA+**	**VDM**	**Z**
Project	Safety-critical	Good?	Med.	**Good**	**Good**	Med.	Good?	**Good**
	Start right away, with help	Med.	**Good**	Med.	Med.	**Good**	Med.	Med.
	Start right away, with initial help	Bad?	Med.	Bad	Bad	**Good**	Med.?	Bad
	Time pressure	-	**Good**	Bad	Bad	**Good**	-	Bad
	Agile setting	Med.?	Med.?	Med.?	Med.?	Good?	Med.?	Med.?
Company	Big	Med.	**Good**	Med.	Med.	**Good**	Med.	Med.
	SME	Bad	**Good**	Bad	Bad	**Good**	Bad	Bad
	cannot afford transition phase	Bad	Med.	Bad	Bad	Med.	Bad	Bad
Goal	Improve product quality	**Good**	**Good**	**Good**	**Good**	**Good**	**Good**	**Good**
	Improve process quality	Med.	**Good**	Med.	Med.	Med.	Med.	Med.
	Reduce specification errors	Med.?	Med.	**Good**	**Good**	**Good**	Med.?	Med.?
	Improve requirements definition	Med.?	**Good**	Med.	Med.	Med.?	Med.?	Med?
	Improve documentation	Med.?	**Good**	Med.	Med.	Good?	Med.?	Med.?
	Improve understanding of design	-	Med.	Med.?	Bad	Med.?	Good?	Med.?
	Foundation for maintenance	Poor	Med.	Med.?	Med.?	Bad?	Med.?	Bad?
	Explore abstract model	Med.?	**Good**	**Good**	**Good**	**Good**	Med.?	Good?
	Basis for test cases	Med	Med.	Med.?	Bad	Med.?	Med.?	**Good**
	Meet safety requirements	Good?	Med.	**Good**	**Good**	Good?	Good?	**Good**

Table 6: Simplified table for project-specific assessment of formal methods

ods, or rigorous methods, in industry, as practitioners and managers should now find it easier to assess the possible impacts of introducing such methods in real-life projects and to select the best suitable methods for their needs.

Most of the criteria were assembled from a structured literature study, supplemented by own experience, whereby we tried to put a special focus on sources close to industry. We came up with five categories into which to sort the criteria, which focus on different aspects to enable more focused assessments. Thereby also a certain amount of redundancy was retained so as to enable assessments based on one or two categories of interest only.

To put these criteria on which we finally settled into practice, we made an assessment of seven particular formal methods, based on the available literature and documentation as well as, in certain cases, own experience. The selected methods are so-called state-based methods and are widely used in industry; however, we hope that others will venture to apply those criteria to other methods as well and we would be interested to hear from such assessments, from corrections of the particular assessments given here, or from alternative assessments.

We concluded our main part with a simplified assessment table for practitioners, allowing for project- or company-specific considerations. However, we found it very hard to securely and reproducibly base our own assessments there on the facts gathered and presented in the previous sections, and the entries there are to be taken with caution. To find a method to base assessments regarding such pragmatic criteria as those in the last table on available facts in a justifiable way will be an interesting challenge for future work.

References

[AAB+13] Pablo Abad, Nazareno Aguirre, Valeria Bengolea, Daniel Ciolek, Marcelo Frias, Juan Galeotti, Tom Maibaum, Mariano Moscato, Nicolás Rosner, and Ignacio Vissani. Improving test generation under rich contracts by tight bounds and incremental sat solving. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pages 21–30, 2013.

[ABH+10] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International journal on software tools for technology transfer*, 12(6):447–466, 2010.

[Abr96] Jean-Raymond Abrial. *The B Book*. Cambridge University Press, 1996.

[Abr03] Jean-Raymond Abrial. Event based sequential program development: Application to constructing a pointer program. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *LNCS*, pages 51–74, 2003.

[Abr10] Jean-Raymond Abrial. *Modeling in Event-B System and Software Design*. Cambridge University Press, Cambridge, 2010.

[ACJ+96] Mark A Ardis, John A Chaves, Lalita Jategaonkar Jagadeesan, Peter Mataga, Carlos Puchol, Mark G Staskauskas, and James Von Olnhausen. A framework for evaluating specification methods for reactive systems. *Software Engineering, IEEE Transactions on*, 22(6):378–389, 1996.

[AFPMdS11] José B. Almeida, Maria J. Frade, Jorge S. Pinto, and Simão Melo de Sousa. *Rigorous Software Development. An Introduction to Program Verification*. Springer, 2011.

[AM98] Jean-Raymond Abrial and Louis Mussat. Introducing Dynamic Constraints in B. In *B '98: Proceedings of the Second International B Conference on Recent Advances in the Development and Use of the B Method*, pages 83–128, London, UK, 1998. Springer-Verlag.

[Anl00] Matthias Anlauff. XASM – an extensible, component-based abstract state machines language. http://xasm.sourceforge.net/XasmAnl00/XasmAnl00.html, last accessed 2015-05-19, March 2000.

[Art11] Rob Arthan. ProofPower. http://www.lemma-one.com/ProofPower/index/ , last updated 2011-02-10, last accessed 2015-06-01, 2011.

[ASM80] Jean-Raymond Abrial, Stephen A. Schuman, and Bertrand Meyer. A specification language. In A.M. Macnaghten and R.M. McKeag, editors, *On the Construction of Programs*. Cambridge University Press, 1980.

[ASR14] Ramadan Abdunabi, Wuliang Sun, and Indrakshi Ray. Enforcing spatio-temporal access control in mobile applications. *Computing*, 96(4):313–353, 2014.

[Ban08] Richard Banach. Model based refinement and the tools of tomorrow. In *1st International Conference of Abstract State Machines, B and Z*, pages 42–56. Springer, 2008.

[Bar98] Milica Barjaktarovic. The state-of-the-art in formal methods. Technical report / Wilkes University and WetStone Technologies,

http://www.cs.utexas.edu/users/csed/formal-methods/docs/StateFM.pdf, last accessed 2015-06-08, 1998.

[BBG+08] Roberto Bruni, Antonio Bucchiarone, Stefania Gnesi, Dan Hirsch, and Alberto Lluch Lafuente. Graph-based design and analysis of dynamic software architectures. In Pierpaolo Degano, Rocco Nicola, and José Meseguer, editors, *Concurrency, Graphs and Models*, pages 37–56. Springer-Verlag, Berlin, Heidelberg, 2008.

[BCF+97] Juan C. Bicarregui, D.L. Clutterbuck, Gavin R. Finnie, Howard P. Haughton, Kevin Lano, H. Lesan, D.W.R.M. Marsh, Brian M. Matthews, Michael R. Moulding, A.Richard Newton, Brian Ritchie, T.G.A. Rushton, and P.N. Scharbach. Formal methods into practice: case studies in the application of the b method. *IEE Proceedings - Software*, 144:119–133(14), April 1997.

[BDSW14] Pontus Boström, Fredrik Degerlund, Kaisa Sere, and Marina Waldén. Derivation of concurrent programs by stepwise scheduling of Event-B models. *Formal Aspects of Computing*, 26(2):281–303, 2014.

[BFJ+11] Jens Bendisposto, Fabian Fritz, Michael Jastram, Michael Leuschel, and Ingo Weigelt. Developing camille, a text editor for rodin. *Software: Practice and Experience*, 41(2):189–198, 2011.

[BFLM05] Jean-Paul Bodeveix, Mamoun Filali, Julia Lawall, and Gilles Muller. Formal methods meet domain specific languages. In Judi Romijn, Graeme Smith, and Jaco van de Pol, editors, *Integrated Formal Methods*, volume 3771 of *Lecture Notes in Computer Science*, pages 187–206. Springer Berlin Heidelberg, 2005.

[BH95] Jonathan P. Bowen and Michael G. Hinchey. Seven more myths of formal methods. *IEEE Softw.*, 12(4):34–41, 1995.

[BH06] Jonathan P. Bowen and Michael G. Hinchey. Ten commandments of formal methods ...ten years later. *Computer*, 39:40–48, 2006.

[Bic98] Juan C. Bicarregui, editor. *Proof in VDM: Case Studies*. Springer-Verlag London, 1998.

[BL96] Peter Baumann and Karl Lermer. Specifying parallel and distributed real-time systems in z. In *Parallel and Distributed Real-Time Systems, 1996. Proceedings of the 4th International Workshop on*, pages 216–222, 1996.

[BL03] Brannon Batson and Leslie Lamport. High-level specifications: Lessons from industry. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Formal Methods for Components and Objects*, volume 2852 of *Lecture Notes in Computer Science*, pages 242–261. Springer Berlin Heidelberg, 2003.

[BM95] Juan C. Bicarregui and Brian M. Matthews. Formal methods in practice: A comparison of two support systems for proof. In *SOFSEM '95: Theory and Practice of Informatics*, volume 1012 of *LNCS*, pages 184–205. Springer, 1995.

[Bou14] Jean-Louis Boulanger, editor. *Formal Methods Applied to Industrial Complex Systems: Implementation of the B Method*. Wiley-ISTE, 2014.

[Bow01] Jonathan P. Bowen. Z: A formal specification notation. In Marc Frappier and Henri Habrias, editors, *Software Specification Methods. An Overview Using a Case Study*, pages 3–19. Springer, London, 2001.

[BRP+14] Julien Brunel, Laurent Rioux, Stéphane Paul, Anthony Faucogney, and Frédérique Vallée. Formal safety and security assessment of an avionic architecture with alloy. In *Third International Workshop on Engineering Safety and Security Systems (ESSS'14)*, pages 8–19, 2014.

[BS03] Egon Börger and Robert Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, Berlin, Heidelberg, 2003.

[BZSH11] Richard Banach, Huibiao Zhu, Wen Su, and Runlei Huang. Formalising the continuous/discrete modeling step. In *Proceedings Refine 2011*, volume 55 of *EPTCS*, pages 121–138, 2011.

[BZSW14] Richard Banach, Huibiao Zhu, Wen Su, and Xiaofeng Wu. A continuous asm modelling approach to pacemaker sensing. *ACM Transactions on Software Engineering and Methodology*, 24(1), September 2014.

[CDE+99] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. The maude system. In Paliath Narendran and Michael Rusinowitch, editors, *Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, pages 240–243. Springer Berlin Heidelberg, 1999.

[CDH+09] Ernie Cohen, Markus Dahlweid, Mark Hillebrand, Dirk Leinenbach, Micha Moskal, Thomas Santen, Wolfram

Schulte, and Stephan Tobies. VCC: A Practical System for Verifying Concurrent C. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 23–42. Springer Berlin Heidelberg, 2009.

[CDLM10] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. Verifying safety properties with the tla+ proof system. In *Automated Reasoning*, pages 142–148. Springer, 2010.

[CGN+05] Colin Campbell, Wolfgang Grieskamp, Lev Nachmanson, Wolfram Schulte, Nikolai Tillmann, and Margus Veanes. Testing concurrent object-oriented systems with spec explorer. In John Fitzgerald, Ian J. Hayes, and Andrzej Tarlecki, editors, *FM 2005: Formal Methods*, volume 3582 of *Lecture Notes in Computer Science*, pages 542–547. Springer Berlin Heidelberg, 2005.

[CK13] Dermot Cochran and Joseph R. Kiniry. Formal model-based validation for tally systems. In *Proceedings of the 4th International Conference on E-Voting and Identity*, Vote-ID'13, pages 41–60, Berlin, Heidelberg, 2013. Springer-Verlag.

[Cle15] *B* Language Reference Manual Version 1.8.7. http://tools.clearsy.com/wp-content/uploads/sites/8/resources/Manrefb_en.pdf , downloaded 2015-08-20, 2015.

[CLH+15] Luís Diogo Couto, Peter Gorm Larsen, Miran Hasanagic, Georgios Kanakis, Kenneth Lausdahl, and Peter W. V. Tran-Jørgensen. Towards enabling overture as a platform for formal notation ides. In *Proceedings Second International Workshop on Formal Integrated Development Environment, F-IDE 2015, Oslo, Norway, June 22, 2015.*, pages 14–27, 2015.

[CWA+96] Edmund M. Clarke, Jeannette M. Wing, Rajeev Alur, Rand Cleaveland, David Dill, Allen Emerson, Stephen Garland, Stephen German, John Guttag, Anthony Hall, Thomas Henzinger, Gerard Holzmann, Cliff Jones, Robert Kurshan, Nancy Leveson, Kenneth McMillan, J Moore, Doron Peled, Amir Pnueli, John Rushby, Natarajan Shankar, Joseph Sifakis, Prasad Sistla, Bernhard Steffen, Pierre Wolper, Jim Woodcock, and Pamela Zave. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4), December 1996.

[DFGV14] David Déharbe, Pascal Fontaine, Yoann Guyot, and Laurent Voisin. Integrating smt solvers in rodin. *Science of Computer Programming*, 94, Part 2:130 – 143, 2014. Abstract State Machines, Alloy, B, VDM, and Z – Selected and extended papers from ABZ 2012.

[DIMS12] Ionut Dinca, Florentin Ipate, Laurentiu Mierla, and Alin Stefanescu. Learn and test for Event-B - A Rodin plugin. In *Abstract State Machines, Alloy, B, VDM, and Z - Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings*, volume 7316 of *Lecture Notes in Computer Science*, pages 361–364. Springer, 2012.

[DKK+14] Petr N. Devyanin, Alexey V. Khoroshilov, Victor V. Kuliamin, Alexander K. Petrenko, and Ilya V. Shchepetkov. Formal verification of OS security model with Alloy and Event-B. In Yamine Ait Ameur and Klaus-Dieter Schewe, editors, *Proc. 4th Int. Conf. on Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2014)*, volume 8477 of *LNCS*, pages 309–313. Springer, 2014.

[Don98] Giovanna Dondossola. Formal methods in the development of safety critical knowledge-based components. In *Proceedings of the KR'98 European Workshop on Validation and Verification of Knowledge- Based Systems*, pages 232–237, 1998.

[DR12] Ashish K. Dwivedi and Santanu K. Rath. Model to specify real time system using z and alloy languages: A comparative approach. In *Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012), International Conference on*, pages 1–6, 2012.

[dSRJ10] Thiago C. de Sousa and Aryldo G. Russo Jr. Starting b specifications from use cases. In Marc Frappier, Uwe Glsser, Sarfraz Khurshid, Rgine Laleau, and Steve Reeves, editors, *Abstract State Machines, Alloy, B and Z*, volume 5977 of *Lecture Notes in Computer Science*, pages 411–411. Springer Berlin Heidelberg, 2010.

[EBM+12] Andrew Edmunds, Michael Butler, Issam Maamria, Renato Silva, and Chris Lovell. Event-B Code Generation: Type Extension with Theories. In John Derrick, John Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene, editors, *Abstract State Machines, Alloy, B, VDM, and Z*, volume 7316 of *Lecture Notes in Computer Science*, pages 365–368. Springer Berlin Heidelberg, 2012.

[Fer08] Ronaldo Rodrigues Ferreira. Automatic code generation and solution estimate for object-oriented embedded software. In *Companion to the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications*, OOPSLA Companion '08, pages 909–910, New York, NY, USA, 2008. ACM.

[FGG07] Roozbeh Farahbod, Vincenzo Gervasi, and Uwe Glässer. CoreASM: An Extensible ASM Execution Engine. *Fundam. Inf.*, 77(1-2):71–103, January 2007.

[FGL+08] Marc Frappier, Frédéric Gervais, Régine Laleau, Benoît Fraikin, and Richard St-Denis. Extending statecharts with process algebra operators. *Innovations in Systems and Software Engineering*, 4(3):285–292, 2008.

[FH06] Marc Frappier and Henri Habrias, editors. *Software Specification Methods*. ISTE, London, 2006.

[FHB+14] Andreas Fürst, Thai Son Hoang, David Basin, Krishnaji Desai, Naoto Sato, and Kunihiko Miyazaki. Code generation for Event-B. In Elvira Albert and Emil Sekerinski, editors, *Integrated Formal Methods*, volume 8739 of *Lecture Notes in Computer Science*, pages 323–338. Springer International Publishing, 2014.

[FHP06] Marc Frappier, Henri Habrias, and Pascal Poizat. A comparison of the specification methods. In Marc Frappier and Henri Habrias, editors, *Software Specification Methods*, pages 353–362. ISTE, London, 2006.

[FLM+05] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs For Object-oriented Systems*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2005.

[FLS08] John Fitzgerald, Peter Gorm Larsen, and Shin Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *SIGPLAN Not.*, 43(2):3–11, Feb 2008.

[FPA04] Marcelo F. Frias, Carlos G. López Pombo, and Nazareno M. Aguirre. An equational calculus for alloy. In Jim Davies, Wolfram Schulte, and Mike Barnett, editors, *Formal Methods and Software Engineering*, volume 3308 of *Lecture Notes in Computer Science*, pages 162–175. Springer Berlin Heidelberg, 2004.

[FSTW15] Flavio Ferrarotti, Klaus-Dieter Schewe, Loredana Tec, and Qing Wang. A new thesis concerning synchronised parallel computing - simplified parallel ASM thesis. *CoRR*, abs/1504.06203, 2015.

[GBF01] Geri Georg, Jores Bieman, and Robert B. France. Using alloy and uml/ocl to specify run-time configuration management: A case study. In *Workshop of the pUML-Group Held Together with the UML'01 on Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists*, pages 128–141. GI, 2001.

[GBL12] Emsaieb Geepalla, Behzad Bordbar, and Joel Last. Transformation of spatio-temporal role based access control specification to alloy. In Alberto Abelló, Ladjel Bellatreche, and Boualem Benatallah, editors, *Model and Data Engineering*, volume 7602 of *Lecture Notes in Computer Science*, pages 67–78. Springer Berlin Heidelberg, 2012.

[GM93] Michael J. C. Gordon and Tom Melham. *Introduction to HOL: A Theorem-Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.

[GMM90] Carlo Ghezzi, Dino Mandrioli, and Angelo Morzenti. TRIO: A Logic Language for Executable Specifications of Real-time Systems. *J. Syst. Softw.*, 12(2):107–123, May 1990.

[GRS08] Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. Model-driven language engineering: The asmeta case study. In *Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on*, pages 373–378, 2008.

[Gur95] Yuri Gurevich. Evolving algebra 1993: Lipari guide. In Egon Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[GZP94] John D. Gannon, Marvin V. Zelkowitz, and James M. Purtilo. *Software Specification: A Comparison of Formal Methods*. Greenwood Publishing, Westpoint, 1994.

[HA10] Thai Son Hoang and Jean-Raymond Abrial. Event-B decomposition for parallel programs. In Marc Frappier, Uwe Glässer, Sarfraz Khurshid, Rgine Laleau, and Steve Reeves, editors, *Abstract State Machines, Alloy, B and Z*, volume 5977 of *Lecture Notes in Computer Science*, pages 319–333. Springer Berlin Heidelberg, 2010.

[Hal90] Anthony Hall. Seven myths of formal methods. *IEEE Software*, 7(5):11–19, September 1990.

[Hau91] Howard P. Haughton. Using z to model and analyse safety and liveness properties of communication proto-

cols. *Information and Software Technology*, 33(8):575 – 580, 1991.

[HJL14] Stefan Hallerstede, Michael Jastram, and Lukas Ladenberger. A method and tool for tracing requirements into specifications. *Sci. Comput. Program.*, 82:2–21, March 2014.

[HL12] Dominik Hansen and Michael Leuschel. Translating tla+ to b for validation with prob. In John Derrick, Stefania Gnesi, Diego Latella, and Helen Treharne, editors, *Integrated Formal Methods*, volume 7321 of *Lecture Notes in Computer Science*, pages 24–38. Springer Berlin Heidelberg, 2012.

[ISO96] ISO. ISO/IEC 13817-1:1996. information technology – programming languages, their environments and system software interfaces – vienna development method – specification language – part 1: Base language. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=22988 , last accessed 2015-06-02, 1996.

[ISO02] ISO. ISO/IEC 13568:2002: Information technology – Z formal specification notation – syntax, type system and semantics. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=21573 , last accessed 2015-06-01, 2002.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[Jac12a] Daniel Jackson. alloy: a language & tool for relational models. http://alloy.mit.edu/alloy/ , last updated 2012, last accessed 2015-06-02, 2012.

[Jac12b] Daniel Jackson. alloy: a language & tool for relational models: applications. http://alloy.mit.edu/alloy/applications.html, last updated 2012, last accessed 2015-12-07, 2012.

[Jac12c] Daniel Jackson. alloy: a language & tool for relational models: frequently asked questions. http://alloy.mit.edu/alloy/faq.html , last updated 2012, last accessed 2015-12-07, 2012.

[Jac12d] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis. 2nd edition*. MIT Press, 2012.

[Jon90] Cliff B. Jones. *Systematic software development using VDM (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[KDGN97] John C Knight, Colleen L DeJong, Matthew S Gibble, and Luís G Nakano. Why are formal methods not used more widely? In *The Fourth NASA Langley Formal Methods Workshop (LFM'97)*, 1997.

[KG06] Jung Soo Kim and David Garlan. Analyzing architectural styles with alloy. In *Proceedings of the ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysis*, ROSATEA '06, pages 70–80, New York, NY, USA, 2006. ACM.

[KGS12] Arvinder Kaur, Samridhi Gulati, and Sarita Singh. Analysis of three formal methods – Z, B and VDM. *International Journal of Engineering Research & Technology (IJERT)*, 1(4), June 2012.

[KIG+15] Felix Kossak, Christa Illibauer, Verena Geist, Jan Kubovy, Christine Natschläger, Thomas Ziebermayr, Theodorich Kopetzky, Bernhard Freudenthaler, and Klaus-Dieter Schewe. *A Rigorous Semantics for BPMN 2.0 Process Diagrams*. Springer, 2015.

[KJ08] Eunsuk Kang and Daniel Jackson. Formal modeling and analysis of a flash filesystem in alloy. In *Abstract State Machines, B and Z*, volume 5238 of *Lecture Notes in Computer Science*, pages 294–308. Springer Berlin Heidelberg, 2008.

[KMGI14] Felix Kossak, Atif Mashkoor, Verena Geist, and Christa Illibauer. Improving the understandability of formal specifications: An experience report. In *Proc. of the 20th Intl. Working Conf. on Requirements Engineering: Foundations for Software Quality REFSQ'14*, volume 8396 of *LNCS*, pages 184–199. Springer, 2014.

[Kos14] Felix Kossak. Landing gear system: An asm-based solution for the abz case study. In Frdric Boniol, Virginie Wiels, Yamine Ait Ameur, and Klaus-Dieter Schewe, editors, *ABZ 2014: The Landing Gear Case Study*, volume 433 of *Communications in Computer and Information Science*, pages 142–147. Springer International Publishing, 2014.

[KYZ+11] Shadi A. Khalek, Guowei Yang, Lingming Zhang, Darko Marinov, and Sarfraz Khurshid. Testera: A tool for testing java programs using alloy specifications. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 608–611, 2011.

[LÏ0] Michael Löwe. Position paper: Formal methods in agile development. *Electronic Communications of the EASST*, 30: Graph and Model Transformation, 2010.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, 1994.

[Lam02] Leslie Lamport. *Specifying Systems. The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[LB08] Michael Leuschel and Michael Butler. ProB: An Automated Analysis Toolset for the B Method. *Journal Software Tools for Technology Transfer*, 10(2):185–203, 2008.

[Lec14] Thierry Lecomte. *Formal Methods Applied to Complex Systems: Implementation of the B Method*, chapter Atelier B, pages 35–45. John Wiley & Sons Inc., 2014.

[LW13] Peter Gorm Larsen and Sune Wolff. Development process of distributed embedded systems using VDM, 2013.

[LYZ97] Xiaodong Liu, Hongji Yang, and Hussein Zedan. Formal methods for the re-engineering of computing systems. In *Proc. 21st Computer Software and Applications Conference (COMPSAC '97)*, pages 409–414, 1997.

[Mas15] Atif Mashkoor. Model-driven development of high-assurance active medical devices. *Software Quality Journal*, pages 1–26, 2015.

[MBD+00] Paul Mukherjee, Fabien Bousquet, Jérôme Delabre, Stephen Paynter, and Peter Gorm Larsen. Exploring timing properties using VDM++ on an industrial application. In *Proceedings of the Second VDM Workshop*, 2000.

[MBDT15] Atif Mashkoor, Miklos Biro, Marton Dolgos, and Peter Timar. Refinement-Based Development of Software-Controlled Safety-Critical Active Medical Devices. In *Software Quality Days 2015*, Lecture Notes in Business Information Processing, pages 120–132. Springer International Publishing Switzerland, 2015.

[McG97] Thomas McGibbon. An analysis of two formal methods: VDM and Z. Technical Report, DoD Data & Analysis Center for Software (DACS), 1997, URL: https://www.csiac.org/sites/default/files/An, last accessed 2015-02-20, 1997.

[Mer08] Stephan Merz. The specification language tla+. In Dines Bjørner and Martin. Henson, editors, *Logics of Specification Languages*, Monographs in Theoretical Computer Science, pages 401–451. Springer Berlin Heidelberg, 2008.

[Mey85] Bertrand Meyer. On formalism in specifications. *Software, IEEE*, 2(1):6–26, Jan 1985.

[MFGL10] Jérémy Milhau, Marc Frappier, Frédéric Gervais, and Régine Laleau. Systematic translation rules from ASTD to event-b. In *Proceedings of the 8th International Conference on Integrated Formal Methods*, IFM'10, pages 245–259, Berlin, Heidelberg, 2010. Springer-Verlag.

[MHB13] Atif Mashkoor, Osman Hasan, and Wolfgang Beer. Using probabilistic analysis for the certification of machine control systems. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics*, volume 8128 of *Lecture Notes in Computer Science*, pages 305–320. Springer Berlin Heidelberg, 2013.

[MHM02] Seyed-Hassan Mirian-HosseinAbadi and Mohammad R. Mousavi. Making nondeterminism explicit in z. In *Proceedings of the Iranian Computer Society Annual Conference (CSICC 02), Tehran, Iran*, 2002.

[MJ10] Atif Mashkoor and Jean-Pierre Jacquot. Domain Engineering with Event-B: Some Lessons We Learned. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 252–261, Sept 2010.

[MJ11] Atif Mashkoor and Jean-Pierre Jacquot. Utilizing Event-B for Domain Engineering: A Critical Analysis. *Requirements Engineering*, 16(3):191–207, 2011.

[MJ15] Atif Mashkoor and Jean-Pierre Jacquot. Observation-Level-Driven Formal Modeling. In *High-Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on*, pages 158–165, 2015.

[MK02] Darko Marinov and Sarfraz Khurshid. Valloy – virtual functions meet a relational language. In Lars-Henrik Eriksson and PeterAlexander Lindsay, editors, *FME 2002:Formal MethodsGetting IT Right*, volume 2391 of *Lecture Notes in Computer Science*, pages 234–251. Springer Berlin Heidelberg, 2002.

[MMFA12] David Mentré, Claude March, Jean-Christophe Fillitre, and Masashi Asuka. Discharging Proof Obligations

from Atelier B Using Multiple Automated Provers. In John Derrick, John Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene, editors, *Abstract State Machines, Alloy, B, VDM, and Z*, volume 7316 of *Lecture Notes in Computer Science*, pages 238–251. Springer Berlin Heidelberg, 2012.

[MMS08] Paulo J. Matos and João Marques-Silva. Model checking Event-B by encoding into Alloy. In Egon Börger, Michael Butler, Jonathan Bowen, and Paul Boca, editors, *Abstract State Machines, B and Z*, volume 5238 of *Lecture Notes in Computer Science*, pages 346–346. Springer Berlin Heidelberg, 2008.

[MRR11] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Semantically configurable consistency analysis for class and object diagrams. In J. Whittle, T. Clark, and T. Kühne, editors, *Proc. 14th Int. Conf. on Model Driven Engineering Languages and Systems (Models 2011)*, volume 6981 of *LNCS*, pages 153–167. Springer, 2011.

[MS11] Dominique Méry and Neeraj Kumar Singh. Automatic code generation from Event-B models. In *Proceedings of the Second Symposium on Information and Communication Technology*, SoICT'11, pages 179–188, New York, NY, USA, 2011. ACM.

[MU05] Petra Malik and Mark Utting. Czt: A framework for z tools. In *ZB 2005: Formal Specification and Development in Z and B*, pages 65–84. Springer, 2005.

[New14] Chris Newcombe. Why Amazon chose TLA+. In *Abstract State Machines, Alloy, B, TLA, VDM, and Z. Proc. 4th Int. Conf. ABZ 2014*, pages 25–39, 2014.

[NRZ+15] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon Web Services uses formal methods. *Communications of the ACM*, 58(4):66–73, April 2015.

[OMG14] OMG. MDA - the architecture of choice for a changing world. http://www.omg.org/mda/ , last updated 2014-08-25, last accessed 2015-04-24, 2014.

[ORA09] ORA Canada. Z/EVES. http://www.oracanada.com/z-eves/welcome.html , last modified 2009-03-28, last accessed 2015-06-01, 2009.

[ORS92] Sam Owre, John M. Rushby, and Natarajan Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.

[Pau94] Lawrence C. Paulson. *Isabelle: a Generic Theorem Prover*. Lecture Notes in Computer Science. Springer – Berlin, 1994.

[PB13] S.K. Pandey and Mona Batra. Formal methods in requirements phase of SDLC. *International Journal of Computer Applications*, 70(13):7–14, May 2013.

[PB15] Richard F. Paige and Phillip J. Brooke. Agile formal method engineering. In *Integrated Formal Methods 2015*, volume 3771 of *LNCS*, pages 109–128, 2015.

[PD06] Christophe Ponsard and Emmanuel Dieul. From requirements models to formal specifications in B. In *An International Workshop on Regulations Modelling and their Validation and Verification - REMO2V'06*, Luxemburg, 2006.

[Ram08] Tahina Ramananandro. Mondex, an electronic purse: specification and refinement checks with the alloy model-finding method. *Formal Aspects of Computing*, 20(1):21–39, 2008.

[RC07] Joris Rehm and Dominique Cansell. Proved Development of the Real-Time Properties of the IEEE 1394 Root Contention Protocol with the Event B Method. In *ISoLA*, pages 179–190, 2007.

[RC14] Víctor Rivera and Néstor Cataño. Translating Event-B to JML-Specified Java programs. In *29th ACM Symposium on Applied Computing, Software Verification and Testing track (SAC-SVT)*, Gyeongju, Korea, March 24-28 2014.

[RdAFLP12] Francisco Rebello de Andrade, João P. Faria, Antónia Lopes, and Ana C.R. Paiva. Specification-driven unit test generation for java generic classes. In John Derrick, Stefania Gnesi, Diego Latella, and Helen Treharne, editors, *Integrated Formal Methods*, volume 7321 of *Lecture Notes in Computer Science*, pages 296–311. Springer Berlin Heidelberg, 2012.

[Rei92] Wolfgang Reif. The kiv system: Systematic construction of verified software. In *Automated DeductionCADE-11*, pages 753–757. Springer, 1992.

[RH11] Marian Rainer-Harbach. Methods and tools for the formal verification of software. an analysis and comparison. Diplomarbeit, Fakultät für Informatik, Technische Universität Wien, URL: https://www.ads.tuwien.ac.at/publications/bib/pdf/rainer-harbach_11.pdf , last accessed 2015-02-23, 2011.

[RH12] Chandan R. Rupakheti and Daqing Hou. Finding errors from reverse-engineered equality models using a constraint solver. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 77–86, 2012.

[RIBQ13] Aurel Randolph, Abdessamad Imine, Hanifa Boucheneb, and Alejandro Quintero. Specification and verification using alloy of optimistic access control for distributed collaborative editors. In Charles Pecheur and Michael Dierkes, editors, *Formal Methods for Industrial Critical Systems*, volume 8187 of *Lecture Notes in Computer Science*, pages 184–198. Springer Berlin Heidelberg, 2013.

[SB06] Colin Snook and Michael Butler. UML-B: Formal modeling and design aided by UML. *ACM Transactions on Software Engineering and Methodology*, 15(1):92–122, 2006.

[SBRG12] Denis Sabatier, Lilian Burdy, Antoine Requet, and Jérôme Guéry. Formal proofs for the nyct line 7 (flushing) modernization project. In John Derrick, John Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene, editors, *Abstract State Machines, Alloy, B, VDM, and Z*, volume 7316 of *Lecture Notes in Computer Science*, pages 369–372. Springer Berlin Heidelberg, 2012.

[Sch11] Matthias Schmalz. Term rewriting in logics of partial functions. In Shengchao Qin and Zongyan Qiu, editors, *Formal Methods and Software Engineering*, volume 6991 of *Lecture Notes in Computer Science*, pages 633–650. Springer Berlin Heidelberg, 2011.

[SD01] Graeme Smith and John Derrick. Specification, refinement and verification of concurrent systems – an integration of Object-Z and CSP. *Formal Methods in System Design*, 18(3):249–284, 2001.

[Sif97] Joseph Sifakis. Formal methods and their evaluation. Position paper presented at FEMSYS in Munich, April, 1997, URL: http://www-verimag.imag.fr/ sifakis/?link=PositionStatements, last accessed 2015-02-24, 1997.

[Sin13] Neeraj Kumar Singh. *Using Event-B for Critical Device Software Systems*. Springer, 2013.

[Spi89] J. Michael Spivey. *The Z notation: a reference manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[SSB01] Robert Stärk, Joachim Schmid, and Egon Börger. *Java and the Java Virtual Machine. Definition, Verification, Validation*. Springer, Berlin, 2001.

[tBBG07] Maurice H. ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Formal methods for service composition. *Annals of Mathematics, Computing & Teleinformatics*, 1(5):1–10, 2007.

[VLH06] Marcel Verhoef, Peter Gorm Larsen, and Jozef Hooman. Modeling and validating distributed embedded real-time systems with VDM++. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 147–162. Springer Berlin Heidelberg, 2006.

[Wan10] Anduo Wang. Formal analysis of network protocols wpe-ii written report. Department of Computer and Information Science, University of Pennsylvania, Philadelphia; http://repository.upenn.edu/cis_reports/924/, last accessed 2016-01-27, 2010.

[WB95] Hélène Waeselynck and Jean-Louis Boulanger. The role of testing in the *b* formal development process. In *Proc. of the 6th Int. Symposium on Software Reliability Engineering*, pages 58–67. IEEE, 1995.

[WJ12] Ting Wang and Dongyao Ji. Active attacking multicast key management protocol using alloy. In *Proceedings of the Third International Conference on Abstract State Machines, Alloy, B, VDM, and Z*, ABZ'12, pages 164–177, Berlin, Heidelberg, 2012. Springer-Verlag.

[WLBF09] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys*, 41(4), 2009.

[Wol08] Burkhart Wolff. Welcome to the Isabelle/HOL-Z home page! https://www.brucker.ch/projects/hol-z/ , last changes 2008-06-16, last accessed 2015-06-01, 2008.

[Wol12] Sune Wolff. Scrum goes formal: Agile methods for safety-critical systems. In *Proc. Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, pages 23–29, 2012.

[Wri09] Steve Wright. Automatic Generation of C from Event-B. In *Workshop on Integration of Model-based Formal Methods and Tools*, 2009.

[WRL14] Xiaoliang Wang, Adrian Rutle, and Yngve Lamo. Scalable verification of model transformations. In Frédéric Boulanger, Michalis Famelis, and Daniel Ratiu, editors, *Proc. 11th Workshop on Model Driven Engineering, Verification and Validation (MoDeVVa 2014)*, pages 29–38, 2014.

[XHM+12] Guo Xie, Xinhong Hei, H. Mochizuki, S. Takahashi, and H. Nakamura. Model based specification validation for automatic train protection and block system. In *Computing and Convergence Technology (ICCCT), 2012 7th International Conference on*, pages 485–488, 2012.

[YML99] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model checking tla+ specifications. In *Correct Hardware Design and Verification Methods*, pages 54–66. Springer, 1999.

[Zav15] Pamela Zave. A practical comparison of Alloy and Spin. *Formal Aspects of Computing*, 2015(2):239–253, March 2015.