Received: 13 March 2018 | Revised: 9 July 2018 | Accepted: 31 July 2018
DOI: 10.1002/spe.2634

EXTENDED CONFERENCE PAPER

WILEY

Evaluating the suitability of state-based formal methods for industrial deployment

Atif Mashkoor[1,2] | Felix Kossak[1] | Alexander Egyed[2]

[1]Software Competence Center Hagenberg GmbH, Hagenberg, Austria
[2]Johannes Kepler University, Linz, Austria

Correspondence
Atif Mashkoor, Software Competence Center Hagenberg GmbH, 4232 Hagenberg, Austria.
Email: atif.mashkoor@scch.at; atif.mashkoor@jku.at

Summary

After a number of success stories in safety-critical domains, we are starting to witness applications of formal methods in contemporary systems and software engineering. However, one thing that is still missing is the evaluation criteria that help software practitioners choose the right formal method for the problem at hand. In this paper, we present the criteria for evaluating and comparing different formal methods. The criteria were chosen through a literature review, discussions with experts from academia and practitioners from industry, and decade-long personal experience with the application of formal methods in industrial and academic projects. The criteria were then evaluated on several model-oriented state-based formal methods. Our research shows that besides technical grounds (eg, modeling capabilities and supported development phases), formal methods should also be evaluated from social and industrial perspectives. We also found out that it is not possible to generate a matrix that renders the selection of the right formal method an automatic process. However, we can generate several pointers, which make this selection process a lot less cumbersome.

KEYWORDS
evaluation criteria, formal methods

1 | INTRODUCTION

After years of advocacy, numerous success stories in the safety-critical systems domain, and the availability of various "easy to use" methods and tools, formal methods are now also being applied to application domains that are not inherently safety critical in nature. One such example is the use of the formal method TLA+[1] at Amazon for specifying and model checking their web services.[2] However, choosing TLA+ was not a straightforward decision for Amazon, and they experimented with other methods before finally selecting one that satisfied their needs. Such experimentations are indeed a nuisance for new adopters of formal methods (and, at times, also for previous adopters). Today, no help is available that navigates practitioners through the intricate process of choosing a formal method suitable for their problem domain. Choosing an inappropriate method for a task leads to undesired results and misconceptions about the method. For example, Amazon initially applied Alloy[3] to the problem, a method that was not particularity designed for this task.

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 needs elaborate evidence for compliance with safety requirements and standards, whereas other projects have budget and time restrictions that do not allow for expensive verification efforts. It also makes a difference whether engineers involved in the project are trained for a particular method and domain or whether their experiences are available for maintenance later on. After all, using formal methods requires skills that most software developers do not have, including familiarity with complex mathematical and logical structures.

Several studies have been published that compare individual formal methods. However, as we will detail in Section 2, many of these studies are either outdated or focus on specific aspects of applying formal methods (often either merely technical criteria of predominantly academic interest or tailored to a particular application domain). To the best of our knowledge, no study has presented evaluation criteria that may help software practitioners in choosing candidate formal methods for their problem at hand by highlighting the methods' strengths and weaknesses.

In this paper, we present a list of criteria for comparing formal methods with respect to general industrial interest. The criteria are drawn from a literature review, discussions with experts and practitioners from academia and industry, and personal experience that stems from the application of formal methods on industrial and academic projects, for example, hemodialysis machines,[4-6] aircraft landing gear system,[7] stereoscopic acuity,[8] transportation systems,[9,10] platooning systems,[11] and business process modeling.[12] In contrast to many other publications, we include a wide range of criteria, which our research and discussions with practitioners from academia and industry yielded to be crucial, for a wider adoption of formal methods in industry. The main motivation behind this paper is to provide the necessary means to help propagate the use of formal methods in day-to-day systems and software engineering.

An earlier version of this paper was presented at the ABZ 2016 conference.[13] In that paper, we mainly presented the criteria for evaluating and comparing different formal methods. This paper also evaluates seven different model-oriented state-based formal methods (ie, Alloy, abstract state machines [ASMs],[14] B,[15] Event-B,[16] TLA+, Vienna Development Method (VDM),[17] and Z[18]) on the stipulated criteria. The evaluation of each method was subsequently checked by one method expert for consistency and completeness. The salient points raised during the discussion on the paper at the ABZ 2016 conference have also been taken into account in this paper.

As we are currently focusing on model-oriented state-based methods and their open-source tools, proprietary tools (and their supported methodologies) like SCADE* (Safety-Critical Application Development Environment) or Simulink† are not considered. Formal methods supporting only specific tasks in the development process like abstract interpretation[19] (effective for detecting runtime errors in software code) or worst-case execution time[20] (useful for calculating tight upper bounds for maximal execution time) are also out of the scope of this paper. Similarly, the evaluation of verification systems, such as PVS,[21] Isabelle,[22] and Coq,[23] is also out of the scope of this paper.

This paper is structured as follows. First, we present our research approach and the list of the reviewed literature in Section 2. Then, in Section 2.3, we present a structured list of criteria for evaluating formal methods and explain why these criteria are important. In Section 3, we compare several model-oriented state-based formal methods with respect to these criteria. This paper is concluded in Section 4, which provides a summary and an outlook for the need of future research.

2 | IDENTIFICATION OF THE CRITERIA

2.1 | Research approach

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

1. What criteria are useful in order to select a candidate 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 literature review complemented by our own experiences with the application of formal methods to industrial and academic projects. The results of our research are then validated by experts from both academia and industry. We limited the search for references by using strings, as follows.

*http://www.esterel-technologies.com/products/scade-suite/
†https://www.mathworks.com/products/simulink.html

- "formal methods" AND "evaluation criteria"
- "formal methods" AND "comparison"
- "formal methods" AND "state of the art"
- "formal methods" AND "literature review"
- "formal methods" AND "take-up"
- "formal methods" AND "barriers"
- "formal methods" AND "industrial application"

We used well-known research repositories and search engines, ie, Google Scholar,[‡] Scopus,[§] IEEE Xplore,[¶] and ACM Digital Library.[#] The bibliographies of the resulting papers also contributed to our research. We further included the literature that we were already aware of or were pointed out by experts. We found (around) 40 relevant papers that we will discuss in the subsequent section.

The search for literature yielded several comparisons between different classical formal methods that were conducted in the 1990s and around 2000. Recently, more comparisons were made in specific settings, typically in the context of university courses. We noted a recent surge in formal method–related tools that can be integrated in traditional development processes. These tools are typically static checkers or model checkers 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, such as by Craigen et al,[24] must certainly be considered outdated, for much has changed since, particularly 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).

After compiling the criteria, we shared it with experts and practitioners from both industry and academia for their feedback. Correspondents from the following selected companies, ranging from small to large enterprises, participated in this study: Alstom (www.alstom.com), Altreonic (www.altreonic.com), ClearSy (www.clearsy.com), Fronius (www.fronius.com), Microsoft (www.microsoft.com), Mitsubishi Electric R&D Center Europe (www.mitsubishielectric-rce.eu), Rockwell Collins (www.rockwellcollins.com), Systerel (www.systerel.fr), and Thales (www.thalesgroup.com). Some of our industrial correspondents had direct experiences with the application of formal methods in their own company, some were consultants who were developing systems for others (and had a very good idea what the overall requirements of their customers were), and some of them were domain experts uninitiated to formal methods. Representatives from the following academic institutes also contributed to the validation of our criteria (though it should be noted that their opinions were mostly about the formal methods of their interest): UC Berkeley (eecs.berkeley.edu), INRIA (www.inria.fr/en/centre/nancy), University of Manchester (www.cs.manchester.ac.uk), MIT (www.csail.mit.edu), University of Newcastle (www.ncl.ac.uk/computing), University of Pisa (di.unipi.it), and University of York (www.cs.york.ac.uk). On the basis of discussions, some elements were added to the list of criteria, and some previous ones were either deleted or further strengthened. The names of all experts who participated in this study are available in the Acknowledgements section of this paper.

2.2 | Literature reviewed

We now present the literature that we found relevant. We observed that the information on concrete evaluations within industry is scarce, though we assume that such evaluations happened. A notable exception is a recent paper by Newcombe on "Why Amazon chose TLA$^+$"[25]; although, it largely only describes experience with TLA+ and, to a lesser extent, Alloy and Microsoft VCC.[26] 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. Two similar feedbacks from industry on the use of formal methods are provided by Schulte[27] and Miller.[28] The former presents a view of systems designers within Microsoft about some aspects of formal methods that can be improved to meet the needs of users (instead of researchers). The latter briefly describes several formal method experiments conducted at Rockwell Collins and attempts to pull the observations together into a profile describing what industry needs from the research community.

[‡]https://scholar.google.com
[§]https://www.scopus.com
[¶]http://ieeexplore.ieee.org
[#] http://dl.acm.org

A position paper from Sifakis[29] also discusses industry-centric evaluation criteria and provides useful input for the list presented in this paper. Also, Sifakis discusses, among others, the crucial point of usability and human factors in general.

The papers from Ardis et al[30] and Knight et al[31] provide frameworks for the evaluation of formal specification languages. The first paper presents criteria and then evaluates several formal languages. The latter paper also presents the perspective of developers, engineers, and computer scientists on these languages.

In "Formal methods: practice and experience," Woodcock et al[32] contribute an overview of historical experiences with formal methods, particularly from 62 industrial projects. The survey was later extended to 98 projects.[33] We could extract several important criteria from these papers, particularly with regard to tool support.

McGibbon[34] discusses different evaluation criteria from a viewpoint of the United States Department of Defense, including more detailed requirements for tools.

Several evaluation criteria can be extracted from the seminal papers by Clarke and Wing[35] and Bowen and Hinchey.[36] The former presents the state of the art and future directions of formal methods, and the latter presents some guidelines to help propagate the use of formal methods in industry. Those criteria can basically all be found in later sources as well.

Liu et al[37] list a number of evaluation criteria and compare a great number of methods. Among others, the authors bring in the additional criterion of applicability in re-engineering, particularly 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; although, unfortunately, much of this information is now (potentially) outdated.

Heitmeyer[38] states that in order to be useful to software practitioners, most of whom lack advanced mathematical training and theorem proving skills, formal methods need to supply user-friendly and easy-to-understand notations, completely automatic (ie, push-button) analysis and feedback tools, and a methodology for integration of formal methods into a standard development process.

An interesting perspective about the application of formal methods is presented by Berry.[39] He states that when formal methods work, they mainly work in the later rounds of development by applying the experience gathered with their application in the earlier rounds and, because of the practitioners' educational background, mentality and willingness to apply formal methods. Berry's point of view is further attested by Robertson.[40]

Banach, in "Model based refinement and the tools of tomorrow,"[41] compares B, Event-B, Z, and the ASM method 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 Gannon et al titled *Software Specification: A Comparison of Formal Methods*[42] focuses on mathematical issues; it discusses only VDM as a formal method in a closer sense, together with temporal logic in general as well as "risk assessment."

Moreover, Kaur et al, in "Analysis of three formal methods - Z, B and VDM,"[43] stress mathematical and modeling issues, but they also mention, eg, tool support, code generation, and testing.

In the book *Software Specification Methods* (ed. by Frappier and Habrias),[44] many different methods are introduced by means of a case study. In the last chapter of the book, 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.

Taylor et al[45] survey several software architecture techniques enabling practitioners to choose the right tool for the job at hand. Rather than focusing on one method, notation, tool, or process, they survey various modeling, design, implementation, deployment, and system adaptation techniques. While putting the elements associated with modeling techniques in context and comparing and contrasting them with one another, they focused on analysis (static analysis, dynamic analysis, performance, etc), refinement, traceability, and design process support (decomposition, distribution, heterogeneity, etc).

A paper by Dondossola[46] specializes on the application domain of safety-critical knowledge-based components and on the method TRIO.[47] Toward 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 Barjaktarovic[48] names requirements for formal methods, particularly 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[49] compare VDM and B based on experience gained in two industrial projects, with a focus on proving.

From an article by Pandey and Batra,[50] we obtained useful assessments of Z and VDM, in particular.

2.3 | Criteria for evaluating formal methods

Now, we will present a structured list of criteria, which is relevant for assessing and comparing formal methods for their usefulness in concrete industrial projects, based on the project's concrete settings. On the basis of discussions with experts, we classify the criteria relevant for industrial projects into five categories. 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.

2.3.1 | Modeling criteria

Modeling criteria concern the scope of systems, the kind of requirements that can be modeled and formalized, and the ease with which such modeling is possible.

The first criterion is *support for composition and decomposition*. Decomposition is important for any application domain when it comes to "larger than toy" systems; without decomposition, large models cannot effectively be overviewed and handled. Decomposition is also of great value in correctness proofs. Composition is important not only as a necessary ingredient to decomposition (gluing decomposed part models together) but also for reuse. Among others, (de)composition is explicitly mentioned as a criterion by Sifakis,[29] Ardis et al,[30] Mcgibbon,[34] Clarke and Wing,[35] Liu et al,[37] Banach,[41] and Kaur et al.[43] This is also an important concern for Thales. A related issue is reuse, which should be supported by appropriate possibilities for parameterization in decomposition. Reuse is explicitly mentioned by Clarke and Wing[35] and Bowen and Hinchey.[36]

The next criterion 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 formal links between models of requirements, design, and implementation can be established within 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 is mentioned as a criterion by Newcombe,[25] Sifakis,[29] Mcgibbon,[34] Liu et al,[37] and Banach.[41] Banach[41] also investigates methods with respect to what notion of refinement they employ, which is relevant for correctness properties. Also, Clarke and Wing[35] mention refinement explicitly, as well as "evolutionary development." Flexible notion of refinement is an important criterion for Thales. The support for the notion of abstraction is equally important[51] and, according to Altrenoic, one of the basic reasons for the use of formal methods.

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 an 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 Dondossola[46]; *concurrency* is mentioned by Newcombe,[25] Mcgibbon,[34] Liu et al,[37] and Kaur et al[43]; and *distributed systems* are mentioned by Newcombe,[25] Sifakis,[29] and Liu et al.[37]

Support for nondeterminism 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 also actually unwanted. Overspecification distracts and impairs overview, 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, such as JeB,[52] offer ways to randomly fill the gaps left by nondeterminism in the model; hence, nondeterminism is not a disadvantage in this respect. Out of the literature reviewed for this work, nondeterminism is mentioned as a criterion only by Liu et al[37]; however, it is implemented in several methods and motivated in the respective method-specific literature.[14]

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, and fairness). Sifakis[29] notes this criterion explicitly. An important class of global properties are reliability properties, which are explicitly mentioned by Mcgibbon[34] and Liu et al,[37] and security policies, which are mentioned by Clarke and Wing.[35]

Support for modeling time must regard sparse and dense models for time separately (see the work of Liu et al[37]). The former is required for general model checking, and the latter is 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 Newcombe,[25] Liu et al,[37] and Dondossola.[46] 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 Newcombe,[25] Sifakis,[29] Mcgibbon,[34] Clarke and Wing,[35] and Liu et al.[37]

Many application domains require that *special concepts* be easily expressed in a modeling language. One important case is hybrid systems that require the modeling of both discrete and continuous state changes (see, eg, the work by Platzer[53]). Hybrid systems are also mentioned in the work of Clarke and Wing.[35] Another example is the modeling of probability, as desired by Clarke and Wing[35] and Liu et al.[37] Liu et al[37] 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[37] separate these issues). Other examples include the modeling of usability properties[34] or of user interface aspects, particularly in safety-critical environments such as interfaces for pilots or air controllers,[34] and queuing theory.[35] Another similar but relatively new phenomenon is the support for cyberphysical systems. Barjaktarovic[48] mentions the criterion of "domain-specific notations for niche markets" in general. The general criterion listed is, of course, domain specific, but existing libraries of reusable 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,"[25] which expresses the last criterion that we adopted in this category. A similar criterion is to be able to introduce custom concepts that are not natively supported by the method. In a related note, Clarke and Wing[35] demand support for sufficient data structures and algorithms.

Additionally, Kaur et al[43] 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. However, a formal method should support different paradigms, eg, object-oriented, procedural, and functional, especially if a formal specification is refined to a program that uses different paradigms. We owe Alstom and Thales for making this explicit.

2.3.2 | Supported development phases

Clarke and Wing[35] 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, particularly for *specification* including *validation* and *verification* of properties.

We think that an unambiguous and analyzable model is vital for any (meaningful) verification attempt because without such a model, it may not be clear what exactly is to be verified against what requirements.

A special phase that 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 the work of Liu et al.[37] A similar criterion is important for Mitsubishi, ie, can the model be extracted automatically from the code?

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.[31,32,34,35,37,43] There are also frequent wishes for *code generation* from formal models[29,30,34,37,43,48] and support for testing.[29-32,34,35,37,43,46,48]

Bug diagnosis is an issue that deserves special mention besides 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 that can be used to identify the problem, but the output is not always easy to use. Newcombe[25] points out the importance of this issue. Barjaktarovic[48] even states that "industry is mostly interested in tools that find bugs rather than tools that prove correctness." Clarke and Wing[35] explicitly mention "counterexamples as a means of debugging." Debugging support is also an important criterion for Mitsubishi.

The potential benefits of formal methods in *maintenance* (as well as reuse) are highlighted by Ardis et al,[30] Knight et al,[31] and Barjaktarovic[48] and endorsed by Thales.

2.3.3 | Technical criteria

In the category of technical criteria, we focus on tool support and how the methods and their 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; hence, the sheer number of available different tools would not be an appropriate criterion. According to Mitsubishi, it is also important to know how much automation a particular tool offers. According to Thales, stability of the tool support is also important.

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 the long-term professional support. Thus, Sifakis[29] brings up the criterion of *commercial support* for tools. Barjaktarovic[48] argues in a similar direction. This was also a major concern for Altreonic.

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, eg, 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 the works of Sifakis[29] and Dondossola,[46] but we think that this should not lead to the conclusion that this issue was of minor importance. Altreonic fully attests to our opinion. Mitsubishi further adds that traceability is an important criterion in general (and not only specific to requirements).

Support for change management addresses the fact that the waterfall model is unrealistic in most settings, ie, 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 as follows: How much stability of the initial specification is presupposed by the method? How easy is it to introduce a change in the specification if the main work has already shifted to design or implementation? How easy is it 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 the work of Dondossola[46] (who states that knowledge-based components can, in practice, only be incrementally specified) and, in a single place, in the work of Liu et al.[37] However, Börger and Stärk[14] stress this issue, and there are even several publications on the use of formal methods within agile development methods.[54]) 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 methods and tools play an important role in the overall development. Although the effect of the use of formal methods on the overall development time is certainly very difficult to measure or assess, studies in the past (see, eg, the work of Woodcock et al[32]) have already shown that the use of formal methods reduces the overall development time. Additionally, as advocated by Berry[39] and as shown by Miller et al,[55] the second-time use of formal methods also reduces the development time markedly. A related and even more general criterion is given by Newcombe[25] as "high return on investment," which includes demands that the method "quickly gives useful results" and "improves time to market." Clarke and Wing[35] note under the keyword of "efficiency" that "turnaround time with an interactive tool should be comparable to that of normal compilation." According to ClearSy, formal methods affect the whole product lifetime and not only the development time because of the approach based on "bug avoidance" instead of "bug fixing."

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 that is required after generation. This criterion is mentioned by Sifakis[29] and Ardis et al.[30] 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 Sifakis.[29] Another important aspect is whether the formal method puts constraints on the kind of system it realizes. For example, SCADE is very good at formalizing reactive systems but much less efficient for state-based systems, by the way memory is handled in its underlying synchronous language. The last point emerged in the discussion with Mitsubishi. According to Alstom, in lieu of mature code generators, implementation guidelines (containing rules for transforming the constructs of a formal notation into equivalent constructs of a given programming language) are the next best thing. For example, it is easy to convert model-based formal specifications, expressed in various notations, into Lisp-like languages and scripting languages like Python. According to Altreonic, compliance and correctness of the generated code is also an important issue to consider.

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. In the cyberphysical era, where software is interacting closely with physical environments and systems are composed of heterogeneous components, the need for interoperability is multifolded. Moreover, interoperability enhances reuse and facilitates technology changes within a company. Knight et al,[31] Clarke and Wing,[35] Banach,[41] and Barjaktarovic[48] explicitly mention this criterion (with a more thorough discussion about the need to combine different methods by the latter). Woodcock et al,[32] among others, describe different projects in which at least two different methods were used to complement each other. Bowen and Hinchey[36] briefly discuss the combination of different formal methods (as well as hybrid methods).

A related criterion is *integration of methodology 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 paper, it is not enough to integrate, for example, 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 Sifakis,[29] Woodcock et al,[32] Clarke and Wing,[35] Bowen and Hinchey,[36] and Barjaktarovic.[48] A similar point was stressed by Systerel and Thales during discussions about the integration of formal methods into existing quality assurance processes and tools of a company.

2.3.4 | Usability

In industrial settings, 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 the work of Kossak et al[56]). 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 Newcombe[25] ("easy to learn and easy to apply," but also "easy to remember"), Sifakis[29] ("time for learning," "ease of learning"), Ardis et al[30] ("learning curve"), Clarke and Wing[35] ("early payback," "incremental gain for incremental effort," "ease of learning"), and Barjaktarovic[48] ("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, and also under *Technical criteria* above, namely, the effect of the method on overall development time.)

General understandability is important because formal models often need to be understood not only by modelers themselves but also, for example, 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 (among others), and maybe even lawyers who want to either defend or to contest whether the respective contract was fulfilled or not (cf the work of Kossak et al[56]). Understandability of requirements specifications is explicitly mentioned by Ardis et al,[30] Dondossola,[46] and Barjaktarovic[48]; related issues are raised by Liu et al[37] (who praise graphical models for being "easy to comprehend") and Mcgibbon[34] (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. Model readability is an important criterion also for Mitsubishi. Altrenoic even calls this a "notational barrier" of 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, ie, 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.[25,29,36,48] Additionally, Systerel reckons that professional trainings are equally important and often demanded by companies. Systerel also thinks that it would be nice if a method provides some methodological (modeling) guidelines like programming guidelines. This will guide beginners on how to use the formalism and also ensures that the whole team uses the formal method in the same way. For Thales, documentation and tutorials are equally important.

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 Knight et al.[31]

2.3.5 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 that do not, or only to a much smaller extent, apply to research projects. Also, as mentioned by Stidolph and Whitehead,[57] 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 students have obtained their doctorate, they may move to some company, potentially a competitor, and even university teachers now usually have only short-term contracts (cf the work of Barjaktarovic[48]). 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 Ardis et al[30] and Bowen and Hinchey.[36]

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 Newcombe[25] and Sifakis.[29] Liu et al[37] and Barjaktarovic[48] mention the state explosion problem as limiting the applicability of, for example, Petri nets to large-scale systems and explicitly note that, for example, Z is "capable for large-scale industrial applications."

Certainly, the actual *amount of industrial experience* that 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, the work of Woodcock et al[32] is a valuable source. The work of Liu et al[37] uses a criterion that is probably largely based on such experience, namely, "industrial strength." Success rate is explicitly listed as a criterion by Sifakis.[29] Also, Liu et al[37] 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 Hall,[58] who claims that the mathematics involved in writing, eg, 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.[‖] According to Alstom, besides a degree in mathematics and foundations in logic, a modeler should also have the right "skills" and "talent." Certainly, style of use matters a lot (cf the work of Kossak et al[56] 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 that is currently available in a particular project setting. The issue of background is raised by Berry[39] and Robertson,[40] among others.

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. Sifakis[29] uses the more general keyword of institutionalization to cover standardization, among other factors for this kind of stability. According to ClearSy, certification of tools is as important as their standardization.

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

[‖] This may be a reflection of the poor state of higher education in computer science, not formal methods.

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).

Last but certainly not least, *previsibility* of resources, efforts, and costs is crucial for the applicability of formal methods to industrial projects. Although it could be (very) difficult to predict and estimate the cost associated with the application of formal methods to a particular task (eg, proving may take a lot of time), this is very important for managerial and executive decisions. Previsibility can benefit from know-how of and familiarity and experience with formal methods and their tool sets. This criterion was explicitly mentioned by almost all of our industrial correspondents.

3 | ASSESSMENT OF SELECTED FORMAL METHODS

3.1 | Overview

We will now use the criteria listed above to compare a range of specific methods that we considered relevant. We clearly favor model-oriented state-based methods because they can be potentially integrated in model-driven engineering. 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.

3.2 | Tables for comparison

We now present a direct comparison between different methods through simplified Tables (see Tables 1 to 5). The rationale for each entry is discussed in the following sections.

"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, "(V. Good)" means very good, 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, and "Adapt." abbreviates adaptable. "Cm." abbreviates "commercial" (licensing), and "OS" means open source. "n/a" means "not applicable."

In Table 1, the support for *refinement* has been adjudged "medium" for some of the methods, like B or Event-B, where refinement plays the pivotal role in their success, or TLA+, where refinement has been an underlaying objective in the design of the method. Our position is that the notion of *n*-to-*m* (*n* abstract models can be refined by *m* concrete ones) refinement, such as featured in ASMs, is more liberal and intuitive for specifying systems at desired levels of abstraction as compared to more restrictive notions of refinements such as one-to-one refinement featured in B. However, one must note that formal methods like B and Event-B indeed excel over other methods when it comes to (mechanically) proving and verifying refinement. As far as a tool-supported proof for preservation of properties in the *n*-to-*m* refinement schema is concerned, it may be a burden, particularly for users who are not specially trained for the task. These observations are already reflected in verification-related entries in Table 2. In Table 1, modeling of *nondeterminism*, *special concepts*, and *rich concepts easily* is omitted because all methods support these concepts up to a certain degree.

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, and *bug diagnoses* is omitted because all of the methods used model checking for this and performed similarly. In Table 3, the criterion of *change management* is omitted due to the lack of information.

TABLE 1 Modeling criteria

	Alloy	ASMs	B	Event-B	TLA+	VDM	Z
(De)Compos.	Y	Med.	Med.	Med.	Y	Y	Good
Refinement	Med.	Good	Med.	Med.	Med.	Med.	(Good)
Parall./concur.	Med.	Good	Part.	N	Good	Y	N
Global propert.	Y	Med.	Med.	Y	Y	N?	Y
Time/perform.	Y	N	(N)	Y	Y	Y	(Y)

TABLE 2 Supported development phases

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

TABLE 3 Technical criteria

	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.
Traceability	Med.	Good	(Y)	Good	-	-	Med.
Time effort	-	Adapt.	(Long)	(Long)	(Short)	-	(Long)
Efficient code	-	n/a	Med.	n/a	n/a	-	n/a
Efficient code gen.	-	(Y)	Y	n/a	n/a	-	n/a
Interoperability	N	(N)	Part.	Part.	Part.	Part.	Y
Integration/IDE	-	(N)	N	(N)	-	(N)	(N)

TABLE 4 Usability

	Alloy	ASMs	B	Event-B	TLA+	VDM	Z
Learning curve	Med.	Good	Med.	Med.	Good	(Good)	Bad
Understandability	Med.	Good	Med.	Med.	Good	Med.	Bad

In Table 4, *documentation* and *support for collaboration* are omitted. The former is omitted because the amount of available documentation is pretty much the same, and the latter is omitted because support for collaboration is generally weak for all the studied methods.

In Table 5, *previsibility* is omitted because it does not depend on the method itself but several factors around the method, eg, familiarity and experience of the staff with the method and the associated tool set.

3.3 | Alloy

Alloy is a state-based method based on "relational algebra, first-order predicate logic, transitive closure, and objects."[59] It is closely related to *Z* (see below), but it has a small and flexible syntax that supports many kinds of modeling styles. It can also be extended to second-order logic to render models fully analyzable. 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. Constraints are described in terms of relations. The powerful automatic analysis and visualization tool, Alloy Analyzer,[3] is a constraint solver (SAT solver).

TABLE 5 Industrial applicability

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

3.3.1 | Modeling criteria

Composition is supported through signature extensions and polymorphic modules. `Merge` (+) and `override` (++) operators can also be used in this regard. Alloy supports the *refinement* mechanism (see, for example, the development of the Mondex system with Alloy[60]). Alloy does support multiple levels of *abstraction* as well as recursively defined relations but not recursive functions. *Concurrent* systems can be modeled in Alloy (see, eg, the work of Brunel et al[61]). As Alloy is based on relational logic and relational calculus, *nondeterminism* is already present.[62] However, this may be a problem because, as also noticed by Zave,[59] nondeterminism can only be implicitly modeled in Alloy. On the other hand, this can be much more powerful than most explicit mechanisms, because one can write a fully relational postcondition, which is not possible in many other methods. Alloy has been used for designing systems where *global system properties of correctness*, such as safety, security, and reliability, have played an important role (see, eg, the work of Brunel et al[61]). Alloy has no direct *notion of time*.[63] However, it is still possible to express timing properties in Alloy models, as demonstrated by Abdunabi et al.[64] Maoz et al[65] selected Alloy for its "expressive power," among others, but according to Newcombe,[25] the *expression of rich concepts* is not easy in Alloy. This is because the Alloy language is minimal, and rich concepts must be expressed idiomatically. Our own impression is that Alloy is a powerful language for expression of rich concepts.

3.3.2 | Supported development phases

Alloy is a system modeling language. It is not exclusively designed for *specification* only, although there have been some specification languages developed based on Alloy. It has been extensively used for the specification of systems such as an electronic purse system[60] and a flash file system.[66] Alloy has also been extensively used for system *verification* and *validation* such as in the domain of distributed collaborative editors[67] and an automatic train protection system.[68] Although Newcombe[25] noted verification as one of the plus points of Alloy, according to Zave,[59] there are certain problems with the automated verification of progress properties with Alloy; part of the proofs had to be performed manually. This is because Alloy uses a bounded exhaustive search to guarantee the absence of counterexamples unlike a deductive approach, and it is intentionally not designed for such tasks. Although, in general, an enumeration-based analysis process may not be suitable for complicated analysis tasks as compared to a technique that is based on deductive reasoning, Alloy makes it possible to analyze very complex formulas without having to have a particular deductive scheme in mind. Validation is supported by the Alloy Analyzer's capability of simulation. The ease of *bug diagnosis* in Alloy is as good as other comparable methods. As Alloy's analysis is bounded and exhaustive, it is excellent at finding bugs quickly, but not quite as good at proving that no bugs exist. However, this is an intentional design decision. The Alloy Analyzer depicts counterexamples as graphs, and such visualizations are indeed helpful. Alloy supports the *design and architecture* of systems.[61] However, according to Kim and Garlan,[69] there are certain problems. 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. However, this is a general problem with verification tools (eg, model checkers) and not specific to Alloy. There is some work about *code generation* from Alloy models such as translation of Alloy models into Java code[70] or compilation of Alloy specifications into implementations that execute against persistent databases.[71] However, the topic of code generation is not very well addressed within the Alloy community. There has been extensive work on using Alloy for *testing* and *automatic test case generation* (see, eg, the works of Abad et al[72] and Khalek et al[73]). However, the aforementioned model-finding limitation is also valid for test case generation. Rupakheti and Hou[74] presented a *reverse engineering* approach 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.

3.3.3 | Technical criteria

The primary *tool support* for Alloy is Alloy Analyzer. In its core, it is a SAT solver that 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 on the Alloy homepage. The Alloy Analyzer** is freely downloadable, including its source code. However, there is no commercial support available for this tool. There are problems regarding *traceability* as, according to Kim and Garlan,[69] it is difficult to relate counterexamples back to the source specification to find what flaw in the design caused it. However, as aforementioned, this is a general problem with verification tools (eg, model checkers) and not specific to Alloy.

**http://alloy.mit.edu/alloy/download.html

3.3.4 | Usability

The *learning curve* is assessed by Newcombe[25] as good "for problems of modest complexity"; from the work of Zave,[59] we infer a medium learning curve. Newcombe[25] presents a relatively bad picture regarding the *understandability* of Alloy models, eg, due to "a significant amount of syntactic overloading" and due to the possibility of combining different notational styles. Maoz et al,[65] on the other hand, selected Alloy for its "readability," among others. Our own impression is that at least more simple models are middling understandable by nonexperts. Actually, Alloy is designed specifically around a simple language and has as few concepts built in as possible, eg, instead of having one way to express concepts like time or state, idioms are used. At times, this may render models difficult to understand, especially for beginners. Newcombe[25] assesses the available *documentation* as good. There is one standard textbook, *Software Abstractions: Logic, Language, and Analysis* by Jackson,[3] and there are links to reference material and tutorials at the Alloy homepage.

3.3.5 | Industrial applicability

Apparently, no *commercial support* is available for the Alloy method. *Scalability* problems are documented by Devyanin et al,[75] and Maoz et al[65] even states that "Alloy was not designed to scale." Scalability is also assessed to be bad by Newcombe.[25] Wang et al[76] discuss ways to tackle scalability and performance problems in the verification of Alloy models; among 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 explicitly mentioned in some cases. It is preferable to have *specially trained staff* for using Alloy, although not absolutely necessary. Alloy is not *standardized.* The Analyzer tool is open source.

3.4 | ASMs

ASMs are a state-based method based on the work of Gurevich[77] and further developed to an industrial-strength modeling method by Börger and Stärk[14] (see, in particular, chapter 9 thereof). 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.

3.4.1 | Modeling criteria

Composition and decomposition is judged as medium by Banach[41]; 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. One distinctive feature of the method is the reusability of ASM models. An industrial strength example is the reuse of the Java/JVM models from the work of Stärk et al[78] to model the ECMA/ISO standard of C# and its virtual machine (see the PhD thesis of Fruja[79] for more details). *Refinement* is good, according to Banach[41] as well as judging by our own experience. The ASM method allows for *n*-to-*m* refinement, which provides maximum flexibility and makes abstraction possible (eg, for reverse engineering). Refinement may combine changes of signature (data refinement) as well as changes of control (procedural refinement) (see pp 25, 110ff in the work of Börger and Stärk[14]). ASM refinements can be proven using SMT solvers.[80] ASMs are well-suited for modeling *parallel and concurrent systems* (see the works of Ferrarotti et al[81] and Börger and Schewe[82]). 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 *nondeterminism* 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, eg, safety and liveness properties. There is also no explicit *notion of time* available. Regarding *special concepts*, there is the work of Banach et al regarding the modeling of continuous systems.[83] 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, eg, as macros.

3.4.2 | 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 walk-throughs 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[84] also enables model checking by tools. Also, proof tools including PVS[21] and Isabelle[22] have been used in the past to verify ASM models, but the required effort for this is rather 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. Only one off-the-shelf *code generator* is currently available for ASM models that can generate C++ code for the Arduino platform.[85] However, the refinement mechanism of ASMs can be and has been used right down to programming code (manually), for example, in the FALKO project at Siemens.[86] A *test generator* is available in the Asmeta platform (see, eg, the work of Arcaini and Gargantini[87]). The ASM method has been used for testing several case studies such as a landing gear system[88] and web applications.[89] The ASM method supports a testing mechanism where model-based testing, ie, models generating test data, test oracles, and test suites, can be complemented by runtime monitoring, ie, checking whether a run of a system satisfies or violates a given correctness property. Therefore, we rate ASMs good on this account. 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 toward the code have to be repeated (manually). If required or desired, also the respective proofs have to be redone. As the tool support is currently under development for refinement and coding, we assess the support of the ASM method for maintenance to be "poor."[‡‡] *Reverse engineering* has been successfully performed using ASMs (see, eg, pp 103ff, 349f, 362, etc, in the work of Börger and Stärk[14]).

3.4.3 | Technical criteria

Overall tool support is at least medium. The tool CoreASM,[90] for which an Eclipse plug-in is available, is widely used 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,[91] but support for this has long been discontinued. An open-source spin-off, XASM,[92] does not seem to have been 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 the 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. While this is difficult to measure in practice in general, the FALKO project report[93] states that absolutely no time was spent on the maintenance phase because the system never broke down, thus saving the time and cost of the project. In any case, extra effort associated with ASMs 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 Banach.[41] 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 on. At present, no such interoperability or integration is given.

[††]According to Börger, the ASM method does not make it difficult to mechanically prove theorems. The difficulty comes from the usual additional efforts one needs if, instead of a handwritten mathematical proof, one wants a machine-generated or machine-checked proof.
[‡‡]Börger thinks that assessing the method's ability based on computational tools is not a good idea!

3.4.4 | 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 *generally understandable*. Although mathematical symbols can be used, the language is, by design, text based. *Documentation* for the method itself is good (see, in particular, the work of Börger and Stärk[14]). Documentation for the available tools is less good, however, consisting largely of scientific publications and manuals that lack overview and may even seem incomplete and/or outdated. One exception is CoreASM,[90] whose behavior is documented by an ASM model for the interpreter, but this is rather an exceptional example.

3.4.5 | Industrial applicability

Professional *support for industrial deployment* is left to consulting by and cooperation with academic personnel but is not institutionalized, at least not on a perceptible scale. (However, for example, our own institution, the Software Competence Center Hagenberg (SCCH) in Austria,[§§] provides such support.) *Scalability* has been proven by large-scale projects (see, eg, the works of Börger and Stärk[14] and Stärk et al[78]). 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, eg, the work of Stärk et al[78]); 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.

3.5 | 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 the Atelier B platform.[¶¶]

3.5.1 | Modeling criteria

Composition and decomposition is supported by the possibility to call operations of other machines and to access, eg, 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 machines. Every machine has its own file. According to Banach,[41] "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 the work of Banach[41]). 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 that may initially be left abstract. B does support *parallelism*, except for code generation, but it does not support *concurrency*.[37,43] 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, eg, temporal properties, as B has no explicit means for the modeling of *timing* or *temporal properties* (see also the work of Liu et al[37]). An extension of the method has been proposed by Abrial and Mussat[94] in this direction. *Reliability* properties can be expressed via invariants, and some reliability properties can moreover be checked via the model checker ProB.[95] 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.

3.5.2 | Supported development phases

According to our own experience (but also according to the work of Woodcock et al[32]), 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

[§§]http://www.scch.at
[¶¶]http://www.atelierb.eu

for *verification*. Atelier B uses an axiomatic proof system (cf the work of Liu et al[37]). ProB adds the possibility of model checking. Regarding *bug diagnosis*, the ProB model checker provides a counterexample with a respective trace. According to Leuschel and Butler,[95] "[...] if ProB finds a counterexample, the user gets important feedback: the proof obligation cannot be discharged, along with a reason why." According to Kaur et al,[43] 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 postprocessing, it is a good basis for the implementation of a B specification. Support for generating test cases is available for B through ProB as demonstrated by Satpathy et al[96] and mentioned by Liu et al.[37] B has been successfully used for *reverse engineering* in the railway domain[##]; 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.

3.5.3 | 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 Ponsard and Dieul.[97] 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, postprocessing 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).

3.5.4 | 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 that are not familiar to most nonmathematicians, 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, 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. *Documentation* is good. There is the B-book,[15] and Atelier B comes with extensive documentation for all its features. The work presented by Bicarregui et al[98] describes several related case studies.

3.5.5 | Industrial applicability

Professional support for *industrial deployment* of the B method is provided by several companies such as ClearSy,[‖] Systerel,[***] 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 cannot be automatically discharged anymore. The same holds if complex data structures (eg, 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; Boulanger[99] mentions several such projects. Liu et al[37] attest that B enjoys "great

[##]http://www.data-validation.fr/data-validation-reverse-engineering/
[‖]http://www.clearsy.com
[***]http://www.systerel.fr

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*[15] and in a freely available reference manual.[†††] For the tool Atelier B, there is a *commercial license* (with support) as well as a free license (with some restrictions, especially no code generators other than for C) available.

3.6 | Event-B

Event-B 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,[100] a platform that supports the writing and proving of specifications.

3.6.1 | Modeling criteria

Regarding *composition and decomposition*, Banach[41] assesses Event-B as "good." However, while applying it to develop a real-life safety-critical system,[4,5] 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. (As an improvement over classical B, an abstract event can be refined by multiple events in the refining machine.) Several techniques have been proposed to liberalize this linear refinement process, eg, retrenchment[101] or observation-level-driven formal modeling.[11] 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[102]). However, both parallel (cf the work of Hoang and Abrial[103]) and concurrent (cf the work of Boström et al[104]) 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 (eg, $:\mid$ or $:\in$) 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 (see, eg, the works of Abrial and Mussat[94] and Rehm[105]) to express such properties in Event-B specifications. Regarding *special concepts*, there exist proposals regarding hybrid and continuous systems (see, in particular, the work of Banach et al[106]). Regarding the *easy expression of rich concepts*, Event-B provides a rich language for set theory and, in particular, relation (and function) theory.

3.6.2 | Supported development phases

Event-B is certainly well suited for formal *specification*. Singh[107] 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,[108] Isabelle/HOL for Rodin,[109] SMT solvers for Rodin,[110] and the model checker ProB. Regarding *bug diagnosis*, the ProB model checker provides a counterexample with a respective trace. Gibson et al[111] report the use of the Event-B method for the engineering of a distributed e-voting system architecture that suggests that this method supports the *design and architecture* phase. A couple of *code generators* have been developed for Event-B. However, of the four tools we found for generating C code, one[112] is not publicly available, one[113] was custom-built and only covers a part of Event-B syntax, EB2ALL[114] explicitly requires manual postprocessing, and Tasking Event-B[115] came out of an academic project that is discontinued. All in all, code generation for Event-B does not appear to be mature and well supported. The MBT (Model-Based Testing) plug-in,[116] which 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. The model checker ProB, on the other hand, provides better results in this context.[117] We do not know of any special support for *maintenance* by some tool for Event-B. Certainly, if code generation works, then 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 reuse of proofs) until regeneration of code becomes possible, but this is possible with other methods as well. The Event-B method has been used for *reverse engineering* of Java/Swing User Interfaces.[118]

[†††]http://tools.clearsy.com/resources/documents

3.6.3 | 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 (eg, Camille[119]), different provers and model checkers (eg, ProB), a requirements traceability tool (eg, ProR[120]), simulation and visualization tools (eg, JeB[52]), 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. Matos et al[121] present an approach for model checking Event-B specifications by their encoding into Alloy. Rivera and Cataño[122] present an approach for translating Event-B to JML. Milhau et al[123] present an approach for translating Algebraic State Transition Diagrams (ASTD) into Event-B. Snook and Butler[124] provide a UML-like graphical front end for Event-B. *Integration in development environments* may be possible relatively easily as Rodin is a plug-in for the widely used Eclipse IDE, but no concrete efforts in that direction are known to us.

3.6.4 | 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 nonmathematicians, 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 since the method and the tool Rodin force one to do so. 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. *Documentation* is good. There is the Event-B book,[16] but on the Event-B and Rodin homepage,[‡‡‡] 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 the Event-B/Rodin homepage), which enables the use of SVN or a similar version-control tool.

3.6.5 | 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 like composition suffer from certain limitations.[5] In comparison with classical B, proof obligations tend to stay simple even as the model grows. Event-B, although popular in industry, is mostly used for the modeling of control systems.[25] Some of the examples of application of Event-B to industrial problems are medical systems[125] and business information systems.[126] 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 Event-B. It is available under an open-source license, as are most of the tools.

3.7 | TLA+

TLA+ is a state-based specification language whose semantics are based on mathematical (Zermelo-Fraenkel) set theory and on Temporal Logic of Actions (TLA).[127] It has been designed by Leslie Lamport when he was working at Digital Equipment Corporation (later bought by Compaq) for the specification of concurrent systems in particular. One of TLA+'s

[‡‡‡]http://www.event-b.org/

major advantages, as described by Lamport, is that it is mostly simple math that one learns in school. The exception is its use of temporal logic; however, that forms only a small part of ordinary specifications.

3.7.1 | Modeling criteria

TLA+ supports different mechanisms for *composition* (see chapter 10 in the work of Lamport[1]) such as through stuttering.[128] Invariance of TLA formulas under finite stuttering is also fundamental for refinement. *Refinement* is assessed as "good" by Newcombe,[25(p28)] and according to Merz,[128(p445)] "A distinctive feature of TLA is its attention to refinement and composition." However, as there is no direct support for refinement in TLA+, we rate it medium (although one can define refinement, for example, through the `INSTANCE` statement (as substitution) and prove refinement, for example, by proving the following property: $Init\_concrete \wedge \Box\,[Next\_concrete] \Rightarrow Init\_abstract \wedge \Box\,[Next\_abstract]$.) 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 Newcombe.[25(p36)] As for *nondeterminism*, Lamport describes an example of a nondeterministic specification through the `CHOOSE` operator (Hilbert's ϵ-operator).[1] Nondeterminism can also be expressed using disjunction and existential quantification, and using PlusCal,[129] which is an algorithmic language based on TLA+. TLA+ does not formally distinguish between specifications and *system properties*: both are written as logical formulas, and concepts such as refinement and composition.[128] The implication $S \Rightarrow I$ can be read as expressing (i) that specification S satisfies property I, (ii) that S refines the abstract specification I, or (iii) that property S is stronger than property I. TLA+ uses set theoretic 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)[130] and the TLC model checker.[131] It is also very common in TLA+ to express correctness through a high-level state machine and to check that a lower-level system specification refines (implies) it. This style of verification is also supported by the model checker for both safety and liveness. There is no built-in notion of *time* in TLA+. However, a simpler method for specifying discrete time is described in the work of Lamport.[132] There is also a standard library module for specifying real-time systems, which is neither supported by the model checker nor the proof system, but still enables the modeling of *performance properties*. According to Newcombe,[25] *rich concepts can be easily expressed* in TLA+.

3.7.2 | 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.[§§§] TLC is a mature and efficient tool and can be used to verify both safety and liveness properties as well as refinement. However, as expected, it is restricted to model checking finite instances. TLAPS, being an interactive theorem prover, on the other hand, is not restricted to finite instances. However, it has a steep learning curve associated with any such tool and is also currently restricted to verifying safety properties. Regarding *bug diagnosis*, the model checker TLC not only presents counterexamples but also traces for analyzing them, eg, by evaluating a certain expression in every state a posteriori. We rate it medium, so as Newcombe.[25] *Performance checking* is possible (see under *modeling of time* above). TLA+ models provide confidence that the models faithfully 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+ specifications by translating TLA+ to B.[133] It can also be used for automated *test case generation*.

3.7.3 | Technical criteria

There exist a few tools for TLA+ but the *overall tool support* is limited. The TLA+ toolbox, available from the TLA+ homepage,[¶¶¶] is an IDE that is available for free. It comprises an editor, a pretty printer, a model checker (TLC), and an interactive proof tool (TLAPS). Also, the animator and model checker ProB, originally developed for B and later adapted for Event-B, additionally supports TLA+. Newcombe[25] assesses tool support to be good. According to Newcombe et al,[2,25]

[§§§]https://tla.msr-inria.inria.fr/tlaps/content/Home.html
[¶¶¶]https://research.microsoft.com/en-us/um/people/lamport/tla/tla.html

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.

3.7.4 | Usability

The *learning curve* is described by Newcombe[25] to be very short. According to Newcombe,[25] TLA+ specifications are reasonably well *understandable*. The alternative language PlusCal, which can be automatically translated to TLA+, appears to be even more understandable for developers due to its C-style syntax. Regarding *documentation*, there is a good introductory book by Lamport which is available for free.[1] The TLA+ homepage lists some other useful resources as well.

3.7.5 | Industrial applicability

TLA+ is supported by Microsoft Corporation, but we have no indication for *professional deployment support*. However, according to Newcombe,[25] 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 the work of Batson and Lamport.[134] Newcombe[25] states that TLA+ has been used successfully on many projects in industry. To name a few, TLA+ has been successfully used by Compaq,[134] Intel,[134] Amazon,[2] and Microsoft (Lamport is currently employed by Microsoft Corporation). According to Newcombe,[25] no *special staff* is required to use TLA+; normal developers can use it easily. Neither TLA nor TLA+ has been *standardized* by an international organization. TLA+ and its standard tools are open source.

3.8 | VDM

The VDM is one of the oldest formal methods—it was developed in the 1970s at the Vienna laboratory of IBM by a group including Heinz Zemanek, Dines Bjørner, Cliff Jones, and others (the work of Jones[135] provides a detailed account about the development of VDM). VDM has three dialects: 1) VDM-SL,[136] 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++,[137] which extends VDM-SL with features for object-oriented modeling and concurrency; and 3) VDM-RT (VDM Real Time), which extends VDM++ with features for describing real-time computations[138] and distributed systems.[139]

3.8.1 | Modeling criteria

Composition is possible.[37] VDM models can be structured into data types (and those into functions) and modules, whereas 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. Although possible in theory, the practical support for refinement is absent in tools. Therefore, support for the notion of refinement in VDM can be rated as medium. In the work of Larsen et al,[140] the authors show the use of VDM for *distributed* embedded systems and explicitly deal with the challenges of *concurrency* and *real-time systems*. As per our own evaluation, *parallelism* and concurrency are given in this method. VDM has also been used for the modeling and validation of distributed embedded systems.[139] Support for *nondeterminism* is available in all dialects of VDM. There are loose choice operators, such as "let, be, st," that confer nondeterminism when used in operations in VDM-SL, VDM++, and VDM-RT. Regarding the *notion of time*, a technical report[140] describes timing analysis for identifying performance bottlenecks using the VDM tool Overture.[###] The works presented by Mukherjee et al[138] and Verhoef et al[139] also deal with real-time systems. Regarding *special modeling concepts*, Mcgibbon[34] and Pandey and Batra[50] note that VDM does have explicit *exception handling*. However, Liu et al[37] note that there are no communication concepts, and Mcgibbon[34] notes that there is no support for performance, reliability, usability, or user interface modeling. Support for some of these concepts, such as communication, performance, and reliability, has been introduced to the method since then. VDM-RT contains a communication bus concept for message passing between processes. The COMPASS Modeling Language (CML)[141] also links features of Circus and VDM to give a formalism with explicit CSP (Communicating Sequential Processes)-like modeling of communications between concurrent processes. VDM also supports the modeling of continuous elements in embedded and cyberphysical systems.[142]

[###] http://overturetool.org/

3.8.2 | Supported development phases

VDM is first and foremost a *specification* method. Apart from specification, the supporting commercial and open-source tools, such as VDMTools,[143] Overture, VDMPad,[144] and VDMJ,‖‖ allow *validation* and *verification* through proofs and debugging. VDM has been used in the *design* phase of projects.[32,34] Pandey and Batra[50] 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 combinatorial testing.[145] Liu et al[37] state that VDM is unsuitable for *reverse engineering*.

3.8.3 | 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, among 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, among 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, Pandey and Batra[50] 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 by Couto et al[146] 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,**** simulation through ProB, and refinement through Maude.[147]

3.8.4 | 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 Pandey and Batra[50] assess VDM specifications to be "easy to understand"). Sufficient *documentation* is available for both the method and tools, including manuals and tutorials.

3.8.5 | 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 a license for the tools are also available freely. Industrial projects in which VDM was used, with resulting code of up to 197 KLOC,[34] suggest that VDM is *scalable* to a considerable degree. There is a considerable amount of *industrial experience* with VDM (see, eg, the works of Woodcock et al[32] and Mcgibbon[34]). 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-SL is standardized by the International Standardization Organization as ISO/IEC 13817-1:1996.

3.9 | 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.[148] Bowen[149] presents a very comprehensive list of references about the Z method covering all its aspects.

3.9.1 | Modeling criteria

Composition (and decomposition) can be achieved by means of either so-called "schemas"[150] or by means of "promotion."[151] As both structuring mechanisms are supported by well-defined theories, we rate the quality of composition mechanisms of Z as good. Z has a well-developed *refinement* theory: its proof rules for refinement are technically complete, with both forward and backward refinement rules, jointly.[152] However, Banach[41] notes, among others, that "spurious traces, not corresponding to real-world behavior, can be generated." Refinement cannot completely go down

‖‖https://github.com/nickbattle/vdmj
****http://research.microsoft.com/en-us/projects/formula/

to the code level, as Bowen[153] 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 good quality.[††††] Z does not directly support the modeling of *concurrency*.[34,37,50] However, the work presented by Smith and Derrick[155] proposed the integration of Object-Z—an object-oriented extension of the Z method—and CSP for specification, refinement, and verification of concurrent systems. A similar objective can also be achieved using Circus.[156] Baumann and Lermer[157] also proposed a framework for the specification of *parallel* and *distributed* real-time systems. *Nondeterminism* is a well-established explicit concept in Z, because it allows, for example, the disjunction of operational schemas with common pre-states (this means that from such a pre-state, each of the operations might fire, and it cannot be predetermined which one). Moreover, since pre-states and post-states can be related by first-order expressions involving disjunctions, nondeterministic data transformations are straightforward to specify. Our observation is attested by Bowen[153] and further explained in the work of Mirian-HosseinAbadi and Mousavi.[158] *Expression of global system properties* is possible in Z. Safety properties are captured adequately by state invariants and operation precondition calculations, whereas liveness can be explored by reachability analysis, either by theorem proving or by using a tool such as ProZ.[159] An explicit *concept of time* is not given for the standard Z method.[34,37] However, Spivey[160] suggests how the notion of time can be specified and used in Z specifications. Regarding *special concepts*, Mcgibbon[34] and Pandey and Batra[50] note that there is no explicit exception handling and no support for user interface modeling. However, it is worth noting that using schema disjunction in composing total interfaces, one can implicitly model exception handling. The language of Z is certainly rich, but whether it allows for *easy expressions* appears to be questionable.

3.9.2 | Supported development phases

Z is a *specification* language in the first place. It supports *validation* via simulation[153] and *verification* via theorem proving as well as model checking.[153] According to Liu et al,[37] there is a "semiautomatic proof system" available for Z. The possible use of Z in *design* is mentioned in the works of Liu et al,[37] Kaur et al,[43] and Pandey and Batra[50] and is rated "strong" in this respect, although Pandey and Batra[50] qualify their assessment, saying that Z was not supporting all aspects of design. Kaur et al[43] state that no code generator is available for Z, and Pandey and Batra[50] even say that a "specification written using Z notation cannot be used to generate computer source code directly"; however, the 1997 paper of McGibbon[34] says that code generation is supported. We ourselves failed to find a code generator; hence, we conclude that *code generation* is probably not available. Kaur et al[43] write that *test generation* for Z is "strong," and Pandey and Batra[50] write that Z "provides a strong base" for *testing*. Usefulness of Z in *reverse engineering* is good.[37]

3.9.3 | Technical criteria

There is at least medium *tool support* for Z. Tool sets available include the Community Z Tools (CZT)[161]—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.[‡‡‡‡] Another proving tool for Z is z-vimes.[§§§§] High-performance proving is also possible with HOL-Z, which is "a proof environment for Z built as a plug-in of the generic theorem prover Isabelle/HOL."[¶¶¶¶] Moreover,[####] provides some details about the former *commercial tool support* available for Z. Currently, a British company, Lemma 1 Ltd,[‖‖‖‖] 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. Knight et al[31] also presented a similar observation. Following the work of Banach,[41] 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/Isabelle can also be used to verify Z specifications.

[††††]Woodcock thinks that in Z, preconditions are preconditions and not firing conditions, so even if some spurious traces are generated, this is not a problem of the Z refinement method. He further adds that he found refinement mechanisms of Z as "good" when they specified, refined, and proved the Mondex Electronic Purse in Z/Eves.[154]
[‡‡‡‡]http://www.lemma-one.com/ProofPower/index/
[§§§§]http://sourceforge.net/projects/z-vimes/
[¶¶¶¶]https://www.brucker.ch/projects/hol-z/
[####] http://www.oracanada.com/z-eves/welcome.html
[‖‖‖‖]http://www.lemma-one.com/

3.9.4 | Usability

We estimate the *learning curve* for Z to be long. Both Knight et al[31] and Bowen[153] confirm this observation. We rate the general *understandability* of a Z specification to be rather bad. Bowen[153] (among 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, eg, 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. Second, 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; hence, formal specification and explaining natural-language text are likely to drift apart. The problem of understandability is exacerbated by Z's frame problem,[41] which forces specifiers to explicitly state for every variable that maintains its value in a step that it does so, thus unnecessarily cluttering the specification. *Documentation* for Z is good.[153]

3.9.5 | 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 Liu et al.[37] However, Newcombe[25] did not find application examples involving the verification of large systems. According to Woodcock et al[32] and Mcgibbon,[34] there is much *industrial experience* with Z; these sources also provide several examples of application of Z on industrial projects. We are certain that *special staff* is required when using the Z method. This is at least partly confirmed by Bowen,[153] 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. This also settles the *licensing* for the method itself; the currently available tools are open source.

4 | CONCLUSION AND FUTURE WORK

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 methods, 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. Although it is not possible to generate a matrix that renders the selection of the right formal method an automatic process, we can generate several pointers, which make this selection process a lot less cumbersome. For example, consider the case of Amazon as discussed in Section 1. Their key requirements were support for modeling complex concurrent and distributed systems, good tool support, small learning curve, and high return on investment, ie, the effect of the method on the overall development time and minimal training.[25] If we now see the corresponding table entries shown in Section 3.2, we would realize that the TLA+ method could have been the first pick for Amazon without going into trial and error.

The first research question raised earlier in this paper, ie, what criteria are useful in order to select a candidate formal method for a particular setting, has been answered in Section 2.3. The proposed criteria were assembled from the relevant literature, whereby we tried to put a special focus on sources close to industry. The criteria were later supplemented by our own experiences and validated by practitioners and experts from industry and academia. After discussion with experts, 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.

The second research question raised in this paper, ie, why are the criteria important for the evaluation of a particular formal method, has been addressed in Sections 2.3.1-2.3.5. Here, we discussed each element of the criteria list in detail.

To put these criteria on which we finally settled into practice, we made an assessment of seven particular model-oriented formal methods, based on the available literature and documentation, our own experience, and discussions with experts of the method. This answers our last research question raised in this paper, ie, how do particular formal methods fare with respect to these criteria.

The selected methods discussed in this paper are so-called state-based methods and are widely used in industry; however, we hope that others will venture to apply these 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. In the future, we also want to evaluate other formal methods widely used in industry such as PVS and SCADE.

ACKNOWLEDGEMENTS

We would like to thank the following persons for their support, discussion, and feedback from academia: Daniel Jackson (MIT), Eunsuk Kang (Berkeley), Joe Near (Berkeley) (on the Alloy section), Egon Börger (Pisa) (on the ASM section), Richard Banach (Manchester) (on B and Event-B sections), Stephan Merz (INRIA) and Leslie Lamport (Microsoft) (on the TLA+ section), John Fitzgerald (Newcastle) (on the VDM section), and Jim Woodcock (York) (on the Z section). We would also like to extend our gratitude to the following persons from industry for discussion and feedback on criteria for evaluating formal methods: Cydeny Minkowitz (Alstom), Eric Verhulst (Altreonic), Thierry Lecomte (ClearSy), Gerrit Steinöcker (Fronius), David Mentré (Mitsubishi Electric R&D Centre Europe), Michael Dierkes (Rockwell Collins), Laurent Voisin (Systerel), and Klaus Reichl (Thales). This work was partially supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

ORCID

Atif Mashkoor http://orcid.org/0000-0003-1210-5953

REFERENCES

1. Lamport L. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Boston, MA: Addison-Wesley; 2002.
2. Newcombe C, Rath T, Zhang F, Munteanu B, Brooker M, Deardeuff M. How Amazon web services uses formal methods. *Commun ACM*. 2015;58(4):66-73.
3. Jackson D. *Software Abstractions: Logic, Language, and Analysis Press*. Cambridge, MA The MIT Press; 2006.
4. Mashkoor A, Biro M, Dolgos M, Timar P. Refinement-Based Development of Software-Controlled Safety-Critical Active Medical Devices. In: *Software Quality. Software and Systems Quality in Distributed and Mobile Environments: 7th International Conference, SWQD 2015, Vienna, Austria, January 20-23, 2015, Proceedings*. Cham, Switzerland: Springer International Publishing Switzerland; 2015:120-132. *Lecture Notes in Business Information Processing*; vol 200.
5. Mashkoor A. Model-driven development of high-assurance active medical devices. *Softw Qual J*. 2016;24(3):571-596.
6. Arcaini P, Bonfanti S, Gargantini A, Mashkoor A, Riccobene E. Integrating formal methods into medical software development: the ASM approach. *Sci Comput Program*. 2018:148-167.
7. Kossak F. Landing gear system: an ASM-based solution for the ABZ case study. In: *ABZ 2014: The Landing Gear Case Study: Case Study Track, Held at the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z, Toulouse, France, June 2-6, 2014. Proceedings*. Cham, Switzerland: Springer International Publishing; 2014:142-147. *Communications in Computer and Information Science*; vol 433.
8. Arcaini P, Bonfanti S, Gargantini A, Mashkoor A, Riccobene E. Formal validation and verification of a medical software critical component. In: Proceedings of the 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE); 2015; Austin, TX.
9. Mashkoor A, Jacquot J-P. Utilizing Event-B for domain engineering: a critical analysis. *Requir Eng*. 2011;16(3):191-207.
10. Mashkoor A, Jacquot J-P. Domain engineering with Event-B: some lessons we learned. In: Proceedings of the 2010 18th IEEE International Requirements Engineering Conference (RE); 2010; Sydney, Australia.
11. Mashkoor A, Jacquot J-P. Observation-level-driven formal modeling. Paper presented at: 2015 IEEE 16th International Symposium on High Assurance Systems Engineering (HASE); 2015; Daytona Beach Shores, FL.
12. Kossak F, Illibauer C, Geist V, et al. *Hagenberg Business Process Modelling Method*. Cham, Switzerland: Springer International Publishing; 2016.
13. Kossak F, Mashkoor A. How to select the suitable formal method for an industrial application: a survey. In: *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 5th International Conference, ABZ 2016, Linz, Austria, May 23-27, 2016, Proceedings*. Cham, Switzerland: Springer International Publishing; 2016:213-228.
14. Börger E, Stärk RF. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2003.
15. Abrial J-R. *The B-Book: Assigning Programs to Meanings*. Cambridge, UK: Cambridge University Press; 1996.
16. Abrial J-R. *Modeling in Event-B: System and Software Engineering*. Cambridge, UK: Cambridge University Press; 2010.
17. Jones CB. *Systematic Software Development Using VDM*. 2nd ed. Upper Saddle River, NJ: Prentice-Hall Inc; 1990.

18. Spivey JM. *The Z notation: a reference manual*. Upper Saddle River, NJ: Prentice-Hall Inc; 1989.
19. Cousot P, Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL); 1977; Los Angeles, CA.
20. Wilhelm R, Engblom J, Ermedahl A, et al. The worst-case execution-time problem–overview of methods and survey of tools. *ACM Trans Embed Comput Syst*. 2008;7:1-36.
21. Owre S, Rushby JM, Shankar N. PVS: a prototype verification system. In: Kapur D, ed. *Automated Deduction–CADE-11: 11th International Conference on Automated Deduction Saratoga Springs, NY, USA, June 15-18, 1992 Proceedings*. Berlin, Germany: Springer-Verlag; 1992:748-752. *Lecture Notes in Computer Science*; vol 607.
22. Nipkow T, Paulson LC, Wenzel M. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Berlin, Germany: Springer; 2002. *Lecture Notes in Computer Science*; vol 2283.
23. Bertot Y, Castéran P. *Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions*. Berlin, Germany: Springer; 2004.
24. Craigen D, Gerhart S, Ralston T. An international survey of industrial applications of formal methods. In: *Z User Workshop, London 1992: Proceedings of the Seventh Annual Z User Meeting, London 14-15 December 1992*. London, UK: Springer; 1993:1-5.
25. Newcombe C. Why Amazon chose TLA$^+$. In: Ait Ameur Y, Schewe K-D, eds. *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2014:25-39.
26. Cohen E, Dahlweid M, Hillebrand M, et al. VCC: a practical system for verifying concurrent C. In: Berghofer S, Nipkow T, Urban C, Wenzel M, eds. *Theorem Proving in Higher Order Logics: 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2009:23-42. *Lecture Notes in Computer Science*; vol 5674.
27. Schulte W. Why doesn't anyone use formal methods? In: *Integrated Formal Methods Second International Conference, IFM 2000 Dagstuhl Castle, Germany, November 1-3, 2000 Proceedings*. Berlin, Germany: Springer; 2000:297-298.
28. Miller SP. The industrial use of formal methods: was Darwin right? In: Proceedings of the Second IEEE Workshop on Industrial Strength Formal Specification Techniques (WIFT); 1998; Boca Raton, FL.
29. Sifakis J. Formal methods and their evaluation. Paper presented at: FEMSYS; 1997; Munich, Germany.
30. Ardis MA, Chaves JA, Jagadeesan LJ, et al. A framework for evaluating specification methods for reactive systems experience report. *IEEE Trans Softw Eng*. 1996;22(6):378-389.
31. Knight JC, DeJong CL, Gibble MS, Nakano LG. Why are formal methods not used more widely? Paper presented at: The Fourth NASA Langley Formal Methods Workshop (LFM'97); 1997; Hampton, VA.
32. Woodcock J, Larsen PG, Bicarregui J, Fitzgerald J. Formal methods: practice and experience. *ACM Comput Surv*. 2009;41:1-19.
33. Fitzgerald J, Bicarregui J, Larsen PG, Woodcock J. Industrial deployment of formal methods: trends and challenges. In: Romanovsky A, Thomas M, eds. *Industrial Deployment of System Engineering Methods*. Berlin, Germany: Springer Berlin Heidelberg; 2013:123-143.
34. Mcgibbon T. *An Analysis of Two Formal Methods: VDM and Z*. Technical Report. Rome, NY: DoD Data & Analysis Center for Software (DACS); 1997.
35. Clarke EM, Wing JM. Formal methods: state of the art and future directions. *ACM Comput Surv*. 1996;28(4):626-643.
36. Bowen JP, Hinchey MG. Ten commandments of formal methods. *Computer*. 1995;28(4):56-63.
37. Liu X, Yang H, Zedan H. Formal methods for the re-engineering of computing systems: a comparison. In: Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC'97); 1997; Atlanta, GA.
38. Heitmeyer C. On the need for *practical* formal methods. In: Ravn AP, Rischel H, eds. *Formal Techniques in Real-Time and Fault-Tolerant Systems: 5th International Symposium FTRTFT'98 Lyngby, Denmark, September 14-18, 1998 Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 1998:18-26.
39. Berry DM. Formal methods: the very idea: some thoughts about why they work when they work. *Sci Comput Program*. 2002;42(1):11-27.
40. Robertson D. Pitfalls of formality in early system design. *Sci Comput Program*. 2002;42(1):29-38.
41. Banach R. Model based refinement and the tools of tomorrow. In: *Abstract State Machines, B and Z: First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings*. Berlin, Germany: Springer; 2008:42-56.
42. Gannon JD, Zelkowitz MV, Purtilo JM. *Software Specification: A Comparison of Formal Methods*. Norwood, NJ: Ablex Publishing; 1994.
43. Kaur A, Gulati S, Singh S. Analysis of three formal methods - Z, B and VDM. *Int J Eng Res Technol*. 2012;1(4):1-4.
44. Frappier M, Habrias H. *Software Specification Methods*. London, UK: ISTE; 2006.
45. Taylor RN, Medvidović N, Dashofy EM. *Software Architecture: Foundations, Theory, and Practice*: Wiley Publishing; 2009.
46. Dondossola G. 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; 1998; Trento, Italy.
47. Ghezzi C, Mandrioli D, Morzenti A. TRIO: a logic Language for executable specifications of real-time systems. *J Syst Softw*. 1990;12(2):107-123.
48. Barjaktarovic M. Wilkes University, WetStone Technologies. *The State-of-The-Art in Formal Methods*. Technical report. 1998.
49. Bicarregui JC, Matthews BM. Formal methods in practice: a comparison of two support systems for proof. In: *SOFSEM '95: Theory and Practice of Informatics: 22nd Seminar on Current Trends in Theory and Practice of Informatics Milovy, Czech Republic November 23 - December 1, 1995 Proceedings*. Berlin, Germany: Springer; 1995:184-205. *Lecture Notes in Computer Science*; vol 1012.
50. Pandey S, Batra M. Formal methods in requirements phase of SDLC. *Int J Comput Appl Technol*. 2013;70(13):7-14.

51. Kramer J. Is abstraction the key to computing? *Commun ACM*. 2007;50(4):36-42.

52. Mashkoor A, Yang F, Jacquot J-P. Refinement-based validation of Event-B specifications. *Softw Syst Model*. 2017;16(3):789-808.

53. Platzer A. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Berlin, Germany: Springer; 2010.

54. Paige RF, Brooke PJ. Agile formal method engineering. In: *Integrated Formal Methods: 5th International Conference, IFM 2005, Eindhoven, The Netherlands, November 29 - December 2, 2005. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2005:109-128. *Lecture Notes in Computer Science*; vol 3771.

55. Miller SP, Greve DA, Srivas MK. Formal verification of the AAMP5 and AAMP-FV microcode. Paper presented at: Third AMAST Workshop on Real-Time Systems; 1996; Salt Lake City, UT.

56. Kossak F, Mashkoor A, Geist V, Illibauer C. Improving the understandability of formal specifications: an experience report. In: *Requirements Engineering: Foundation for Software Quality: 20th International Working Conference, REFSQ 2014, Essen, Germany, April 7-10, 2014. Proceedings*. Cham, Switzerland: Springer; 2014:184-199. *Lecture Notes in Computer Science*; vol 8396.

57. Stidolph DC, Whitehead J. Managerial issues for the consideration and use of formal methods. In: Araki A, Gnesi S, Mandrioli D, eds. *FME 2003: Formal Methods: International Symposium of Formal Methods Europe, Pisa, Italy, September 8-14, 2003. Proceedings*. Berlin, Germany: Springer; 2003:170-186. *Lecture Notes in Computer Science*; vol 2805.

58. Hall A. Seven myths of formal methods. *IEEE Softw*. 1990;7(5):11-19.

59. Zave P. A practical comparison of Alloy and Spin. *Form Asp Comput*. 2015;27(2):239-253.

60. Ramananandro T. *Mondex*, an electronic purse: specification and refinement checks with the *Alloy* model-finding method. *Form Asp Comput*. 2008;20(1):21-39.

61. Brunel J, Rioux L, Paul S, Faucogney A, Vallée F. Formal safety and security assessment of an avionic architecture with Alloy. Paper presented at: 3rd International Workshop on Engineering Safety and Security Systems 2014 (ESSS); 2014; Singapore, Singapore.

62. Marinov D, Khurshid S. VAlloy – virtual functions meet a relational language. In: Eriksson L-H, Lindsay PA, eds. *FME 2002:Formal Methods-Getting IT Right: International Symposium of Formal Methods Europe Copenhagen, Denmark, July 22-24, 2002 Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2002:234-251. *Lecture Notes in Computer Science*; vol 2391.

63. Georg G, Bieman J, France RB. Using Alloy and UML/OCL to specify run-time configuration management: a case study. Paper presented at: Workshop of the pUML-Group Held Together With the UML'01 on Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists; 2001; Toronto, Canada.

64. Abdunabi R, Sun W, Ray I. Enforcing spatio-temporal access control in mobile applications. *Computing*. 2014;96(4):313-353.

65. Maoz S, Ringert JO, Rumpe B. Semantically configurable consistency analysis for class and object diagrams. In: Whittle J, Clark T, Kühne T, eds. *Model Driven Engineering Languages and Systems: 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings*. Berlin, Germany: Springer; 2011. *Lecture Notes in Computer Science*; vol 6981.

66. Kang E, Jackson D. Formal modeling and analysis of a flash filesystem in Alloy. In: *Abstract State Machines, B and Z: First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2008:294-308. *Lecture Notes in Computer Science*; vol 5238.

67. Randolph A, Imine A, Boucheneb H, Quintero A. Specification and verification using Alloy of optimistic access control for distributed collaborative editors. In: Pecheur C, Dierkes M, eds. Formal Methods for Industrial Critical Systems. *Berlin, Germany*. Berlin, Germany: Springer Berlin Heidelberg; 2013:184-198. *Lecture Notes in Computer Science*; vol 8187.

68. Xie G, Hei X, Mochizuki H, Takahashi S, Nakamura H. Model based specification validation for automatic train protection and block system. In: 2012 7th International Conference on Computing and Convergence Technology (ICCCT); 2012; Seoul, South Korea.

69. Kim JS, Garlan D. Analyzing architectural styles with Alloy. In: Proceedings of the ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysis; 2006; Portland, ME.

70. Ferreira RR. 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); 2008; Nashville, TN.

71. Krishnamurthi S, Fisler K, Dougherty DJ, Yoo D. Alchemy: transmuting base Alloy specifications into implementations. In: Proceedings of The 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering; 2008; Atlanta, GA.

72. Abad P, Aguirre N, Bengolea V, et al. Improving test generation under rich contracts by tight bounds and incremental SAT solving. Paper presented at: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation; 2013; Luxembourg City, Luxembourg.

73. Khalek SA, Yang G, Zhang L, Marinov D, Khurshid S. TestEra: a tool for testing Java programs using Alloy specifications. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE); 2011; Lawrence, KS.

74. Rupakheti CR, Hou D. Finding errors from reverse-engineered equality models using a constraint solver. In: Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM); 2012; Trento, Italy.

75. Devyanin PN, Khoroshilov AV, Kuliamin VV, Petrenko AK, Shchepetkov IV. Formal verification of OS security model with Alloy and Event-B. In: Ameur A, Schewe KD, eds. *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings*. Berlin, Germany: Springer; 2014:309-313. *Lecture Notes in Computer Science*; vol 8477.

76. Wang X, Rutle A, Lamo Y. Scalable verification of model transformations. In: Proceedings of the 11th Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa); 2014; Valencia, Spain.

77. Gurevich Y. Evolving algebra 1993: Lipari guide. In: Börger E, ed. *Specification and Validation Methods*. Oxford, UK: Oxford University Press; 1995:9-36.

78. Stärk R, Schmid J, Börger E. *Java and the Java Virtual Machine: Definition, Verification, Validation*. Berlin, Germany: Springer; 2001.

79. Fruja NG. *Type Safety of C# and .NET CLR* [PhD dissertation]. Zürich, Switzerland: ETH Zürich 2007.
80. Arcaini P, Gargantini A, Riccobene E. SMT-based automatic proof of ASM model refinement. In: De Nicola R, Kühn E, eds. *Software Engineering and Formal Methods: 14th International Conference, SEFM 2016, Held as Part of STAF 2016, Vienna, Austria, July 4-8, 2016, Proceedings*. Cham, Switzerland: Springer International Publishing; 2016:253-269.
81. Ferrarotti F, Schewe K-D, Tec L, Wang Q. A new thesis concerning synchronised parallel computing - simplified parallel ASM thesis. *Theor Comput Sci*. 2016;649:25-53.
82. Börger E, Schewe K-D. Concurrent abstract state machines. *Acta Informatica*. 2016;53(5):469-492.
83. Banach R, Zhu H, Su W, Wu X. A continuous ASM modelling approach to pacemaker sensing. *ACM Trans Softw Eng Methodol*. 2014;24(1):1-40.
84. Gargantini A, Riccobene E, Scandurra P. Model-driven language engineering: the ASMETA case study. In: Proceedings of the 2008 The Third International Conference on Software Engineering Advances (ICSEA); 2008; Sliema, Malta.
85. Bonfanti S, Carissoni M, Gargantini A, Mashkoor A. Asm2C++: a tool for code generation from abstract state machines to Arduino. In: Barrett C, Davies M, Kahsai T, eds. *NASA Formal Methods: 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*. Cham, Switzerland: Springer International Publishing; 2017:295-301.
86. Schmid J. Compiling abstract state machines to C++. *J Univers Comput Sci*. 2001;7(11):1069-1088.
87. Arcaini P, Gargantini A. Test generation for sequential nets of abstract state machines with information passing. *Sci Comput Program*. 2014;94:93-108.
88. Arcaini P, Gargantini A, Riccobene E. Offline model-based testing and runtime monitoring of the sensor voting module. In: Boniol F, Wiels V, Ait Ameur Y, Schewe K-D, eds. *ABZ 2014: The Landing Gear Case Study: Case Study Track, Held at the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z, Toulouse, France, June 2-6, 2014. Proceedings*. Cham, Switzerland: Springer International Publishing; 2014:95-109.
89. Bolis F, Gargantini A, Guarnieri M, Magri E, Musto L. Model-driven testing for web applications using abstract state machines. In: *Current Trends in Web Engineering: ICWE 2012 International Workshops: MDWE, ComposableWeb, WeRE, QWE, and Doctoral Consortium, Berlin, Germany, July 23-27, 2012, Revised Selected Papers*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2012:71-78.
90. Farahbod R, Gervasi V, Glässer U. CoreASM: an extensible ASM execution engine. *Fundam Informaticae*. 2007;77(1-2):71-103.
91. Campbell C, Grieskamp W, Nachmanson L, Schulte W, Tillmann N, Veanes M. Testing concurrent object-oriented systems with spec explorer. In: Fitzgerald J, Hayes IJ, Tarlecki A, eds. *FM 2005: Formal Methods: International Symposium of Formal Methods Europe, Newcastle, UK, July 18-22, 2005. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2005:542-547. *Lecture Notes in Computer Science*; vol 3582.
92. Anlauff M. XASM-an extensible, component-based abstract state machines language. In: Gurevich Y, Kutter PW, Odersky M, Thiele L, eds. *Abstract State Machines - Theory and Applications: International Workshop, ASM 2000 Monte Verità, Switzerland, March 19-24, 2000 Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2000:69-90.
93. Börger E, Päppinghaus P, Schmid J. Report on a practical application of ASMs in software design. In: Gurevich Y, Kutter PW, Odersky M, Thiele L, eds. *Abstract State Machines - Theory and Applications: International Workshop, ASM 2000 Monte Verità, Switzerland, March 19-24, 2000 Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2000:361-366.
94. Abrial J-R, Mussat L. Introducing dynamic constraints in B. In: *B'98: Recent Advances in the Development and Use of the B Method: Second International B Conference Montpellier, France, April 22-24, 1998 Proceedings*. London, UK: Springer-Verlag; 1998:83-128.
95. Leuschel M, Butler M. ProB: an automated analysis toolset for the B method. *Int J Softw Tools Technol Transfer*. 2008;10(2):185-203.
96. Satpathy M, Butler M, Leuschel M, Ramesh S. Automatic testing from formal specifications. In: *Tests and Proofs First International Conference, TAP 2007, Zurich, Switzerland, February 12-13, 2007. Revised Papers*. Berlin, Germany: Springer-Verlag; 2007:95-113.
97. Ponsard C, Dieul E. From requirements models to formal specifications in B. Paper presented at: International Workshop on Regulations Modelling and their Validation and Verification - REMO2V'06; 2006; Luxemburg City, Luxemburg.
98. Bicarregui JC, Clutterbuck DL, Finnie G, et al. Formal methods into practice: case studies in the application of the B method. *IEE Proc Softw*. 1997;144:119-133.
99. Boulanger J-L. *Formal Methods Applied to Industrial Complex Systems: Implementation of the B Method*. London, UK: ISTE; 2014.
100. Abrial J-R, Butler M, Hallerstede S, Hoang TS, Mehta F, Voisin L. Rodin: an open toolset for modelling and reasoning in Event-B. *Int J Softw Tools Technol Trans*. 2010;12(6):447-466.
101. Banach R, Poppleton M. Retrenchment: an engineering variation on refinement. In: *B'98: Recent Advances in the Development and Use of the B Method: Second International B Conference Montpellier, France, April 22-24, 1998 Proceedings*. Berlin, Germany: Springer-Verlag; 1998:129-147.
102. Abrial J-R. Event based sequential program development: application to constructing a pointer program. In: Araki K, Gnesi S, Mandrioli D, eds. *FME 2003: Formal Methods International Symposium of Formal Methods Europe, Pisa, Italy, September 8-14, 2003. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2003:51-74. *Lecture Notes in Computer Science*; vol 2805.
103. Hoang T, Abrial J-R. Event-B decomposition for parallel programs. In: Frappier M, Glässer U, Khurshid S, Laleau R, Reeves S, eds. *Abstract State Machines, Alloy, B and Z: Second International Conference, ABZ 2010, Orford, QC, Canada, February 22-25, 2010. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2010:319-333. *Lecture Notes in Computer Science*; vol 5977.
104. Boström P, Degerlund F, Sere K, Waldén M. Derivation of concurrent programs by stepwise scheduling of Event-B models. *Form Asp Comput*. 2014;26(2):281-303.
105. Rehm J. Proved development of the real-time properties of the IEEE 1394 root contention protocol with the Event-B method. *Int J Softw Tools Technol Transfer*. 2010;12(1):39-51.

106. Banach R, Zhu H, Su W, Huang R. Formalising the continuous/discrete modeling step. In: *EPTCS 55: Proceedings 15th International: Refinement Workshop: Limerick, Ireland, 20th June 2011*. Open Publishing Association; 2011:121.

107. Singh NK. *Using Event-B for Critical Device Software Systems*. London, UK: Springer; 2013.

108. Mentré D, Marché C, Filliâtre J-C, Asuka M. Discharging proof obligations from Atelier B using multiple automated provers. In: Derrick J, Fitzgerald J, Gnesi S, et al, eds. *Abstract State Machines, Alloy, B, VDM, and Z: Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2012:238-251. *Lecture Notes in Computer Science*; vol 7316.

109. Schmalz M. Term rewriting in logics of partial functions. In: Qin S, Qiu Z, eds. *Formal Methods and Software Engineering: 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26-28, 2011. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2011:633-650. *Lecture Notes in Computer Science*; vol 6991.

110. Déharbe D, Fontaine P, Guyot Y, Voisin L. Integrating SMT solvers in Rodin. *Sci Comput Program*. 2014;94:130-143.

111. Gibson JP, Lallet E, Raffy J-L. Engineering a distributed e-Voting system architecture: meeting critical requirements. In: *Architecting Critical Systems: First International Symposium, ISARCS 2010, Prague, Czech Republic, June 23-25, 2010 Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2010;89-108.

112. Fürst A, Hoang T, Basin D, Desai K, Sato N, Miyazaki K. Code generation for Event-B. In: Albert E, Sekerinski E, eds. *Integrated Formal Methods: 11th International Conference, IFM 2014, Bertinoro, Italy, September 9-11, 2014, Proceedings*. Cham, Switzerland: Springer International Publishing; 2014:323-338. *Lecture Notes in Computer Science*; vol 8739.

113. Wright S. Automatic generation of C from Event-B. Paper presented at: Workshop on Integration of Model-Based Formal Methods and Tools; 2009; Nantes, France.

114. Méry D, Singh NK. Automatic code generation from Event-B models. In: Proceedings of the Second Symposium on Information and Communication Technology (SoICT); 2011; Hanoi, Vietnam.

115. Edmunds A, Butler M, Maamria I, Silva R, Lovell C. Event-B code generation: type extension with theories. In: Derrick J, Fitzgerald J, Gnesi S, et al eds. *Abstract State Machines, Alloy, B, VDM, and Z: Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2012:365-368. *Lecture Notes in Computer Science*; vol 7316.

116. Dinca I, Ipate F, Mierla L, Stefanescu A. 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*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2012:361-364. *Lecture Notes in Computer Science*; vol 7316.

117. Savary A, Frappier M, Leuschel M, Lanet J-L. Model-based robustness testing in Event-B using mutation. In: Calinescu R, Rumpe B, eds. *Software Engineering and Formal Methods: 13th International Conference, SEFM 2015, York, UK, September 7-11, 2015. Proceedings*. Cham, Switzerland: Springer International Publishing; 2015:132-147.

118. Cortier A, d'Ausbourg B, Aït-Ameur Y. Formal validation of Java/swing user interfaces with the Event B method. In: *Human-Computer Interaction. Interaction Design and Usability: 12th International Conference, HCI International 2007, Beijing, China, July 22-27, 2007, Proceedings, Part I*. Berlin, Germany: Springer Berlin Heidelberg; 2007;1062-1071.

119. Bendisposto J, Fritz F, Jastram M, Leuschel M, Weigelt I. Developing Camille, a text editor for Rodin. *Softw Pract Exp*. 2011;41(2):189-198.

120. Hallerstede S, Jastram M, Ladenberger L. A method and tool for tracing requirements into specifications. *Sci Comput Program*. 2014;82:2-21.

121. Matos PJ, Marques-Silva J. Model checking Event-B by encoding into Alloy. In: Börger E, Butler M, Bowen J, Boca P, eds. *Abstract State Machines, B and Z: First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2008:346-346. *Lecture Notes in Computer Science*; vol 5238.

122. Rivera V, Cataño N. Translating Event-B to JML-specified Java programs. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC); 2014; Gyeongju, South Korea.

123. Milhau J, Frappier M, Gervais F, Laleau R. Systematic translation rules from ASTD to Event-B. In: *Integrated Formal Methods: 8th International Conference, IFM 2010, Nancy, France, October 11-14, 2010. Proceedings*. Berlin, Germany: Springer-Verlag; 2010: 245-259.

124. Snook C, Butler M. UML-B: formal modeling and design aided by UML. *ACM Trans Softw Eng Methodol*. 2006;15(1):92-122.

125. Mashkoor A, Biro M. Towards the trustworthy development of active medical devices: a hemodialysis case study. *IEEE Embed Syst Lett*. 2016;8(1):14-17.

126. Bodeveix J-P, Filali M, Lawall J, Muller G. Formal methods meet domain specific languages. In: Romijn J, Smith G, van de Pol J, eds. *Integrated Formal Methods 5th International Conference, IFM 2005, Eindhoven, The Netherlands, November 29 - December 2, 2005. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2005:187-206. *Lecture Notes in Computer Science*; vol 3771.

127. Lamport L. The temporal logic of actions. *ACM Trans Program Lang Syst*. 1994;16(3):872-923.

128. Merz S. The specification language TLA+. In: Bjørner D, Henson M, eds. *Logics of Specification Languages*. Berlin, Germany: Springer Berlin Heidelberg; 2008:401-451. *Monographs in Theoretical Computer Science*.

129. Lamport L. The PlusCal algorithm language. In: *Theoretical Aspects of Computing - ICTAC 2009: 6th International Colloquium, Kuala Lumpur, Malaysia, August 16-20, 2009. Proceedings*. Berlin, Germany: Springer; 2009:36-60.

130. Cousineau D, Doligez D, Lamport L, Merz S, Ricketts D, Vanzetto H. TLA$^+$ proofs. In: Giannakopoulou D, Méry D, eds. *FM 2012: Formal Methods 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*. Berlin, Germany: Springer; 2012:147-154. *Lecture Notes in Computer Science*; vol 7436.

131. Yu Y, Lamport L, Manolios P. Model checking TLA$^+$ specifications. In: *Correct Hardware Design and Verification Methods 10th IFIP WG10.5 Advanced Research Working Conference, CHARM' 99 BadHerrenalb,Germany,September 27-29, 1999 Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 1999:54-66.

132. Lamport L. Real-time model checking is really simple. In: *Correct Hardware Design and Verification Methods: 13th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2005, Saarbrücken, Germany, October 3-6, 2005. Proceedings*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2005:162-175.

133. Hansen D, Leuschel M. Translating TLA$^+$ to B for validation with ProB. In: Derrick J, Gnesi S, Latella D, Treharne H, eds. *Integrated Formal Methods 9th International Conference, IFM 2012, Pisa, Italy, June 18+21, 2012. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2012:24-38. *Lecture Notes in Computer Science*; vol 7321.

134. Batson B, Lamport L. High-level specifications: lessons from industry. In: de Boer FS, Bonsangue MM, Graf S, de Roever W-P, eds. *Formal Methods for Components and Objects: First International Symposium, FMCO 2002, Leiden, The Netherlands, November 5-8, 2002, Revised Lectures*. Berlin, Germany: Springer Berlin Heidelberg; 2003:242-261. *Lecture Notes in Computer Science*; vol 2852.

135. Jones CB. Scientific decisions which characterize VDM. In: Wing JM, Woodcock J, Davies J, eds. *FM'99 - Formal Methods: World Congress on Formal Methods in the Development of Computing Systems Toulouse, France, September 20-24, 1999 Proceedings, Volume I*. Berlin, Germany: Springer Berlin Heidelberg; 1999:28-47.

136. Fitzgerald J, Larsen PG. *Modelling Systems: Practical Tools and Techniques in Software Development*. 2nd ed. New York, NY: Cambridge University Press; 2009.

137. Fitzgerald J, Larsen PG, Mukherjee P, Plat N, Verhoef M. *Validated Designs For Object-Oriented Systems*. London, UK: Springer-Verlag London Limited; 2005.

138. Mukherjee P, Bousquet F, Delabre J, Paynter S, Larsen PG. Exploring timing properties using VDM++ on an industrial application. In: Proceedings of the Second VDM Workshop; 2000.

139. Verhoef M, Larsen PG, Hooman J. Modeling and validating distributed embedded real-time systems with VDM++. In: Misra J, Nipkow T, Sekerinski E, eds. *FM 2006: Formal Methods: 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2006:147-162. *Lecture Notes in Computer Science*; vol 4085.

140. Larsen PG, Wolff S, Battle N, Fitzgerald J, Pierce K. Development process of distributed embedded systems using VDM. Overture-Open-source Tools for Formal Modelling TR-2010-02. 2010.

141. Woodcock J, Bryans J, Canham S, Foster S. The COMPASS modelling language: timed semantics in UTP. In: *Communicating Process Architectures 2014*. Oxfordshire, UK: Open Channel Publishing; 2014:1-32.

142. Fitzgerald J, Gamble C, Larsen PG, Pierce K, Woodcock J. Cyber-physical systems design: formal foundations, methods and integrated tool chains. In: Proceedings of the Third FME Workshop on Formal Methods in Software Engineering (Formalise); 2015; Florence, Italy.

143. Fitzgerald J, Larsen PG, Sahara S. VDMTools: advances in support for formal modeling in VDM. *ACM SIGPLAN Not*. 2008;43(2): 3-11.

144. Oda T, Araki K, Larsen PG. VDMPad: a lightweight IDE for exploratory VDM-SL specification. In: Proceedings of the Third FME Workshop on Formal Methods in Software Engineering (Formalise); 2015; Florence, Italy.

145. Larsen PG, Lausdahl K, Battle N. Combinatorial testing for VDM. In: Proceedings of the 2010 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM); 2010; Pisa, Italy.

146. Couto LD, Larsen PG, Hasanagić M, Kanakis G, Lausdahl K, Tran-Jørgensen PWV. Towards enabling overture as a platform for formal notation IDEs. In: Proceedings of the 2nd International Workshop on Formal Integrated Development Environment (F-IDE); 2015; Oslo, Norway.

147. Clavel M, Durán F, Eker S, et al. The Maude system. In: Narendran P, Rusinowitch M, eds. *Rewriting Techniques and Applications: 10th International Conference, RTA-99 Trento, Italy, July 2-4, 1999 Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 1999:240-243. *Lecture Notes in Computer Science*; vol 1631.

148. Abrial J-R, Schuman SA, Meyer B. Specification language. In: Macnaghten A, McKeag R, eds. *On the Construction of Programs: An Advanced Course*. Cambridge, UK: Cambridge University Press; 1980:343-410.

149. Bowen JP. Select Z Bibliography. In: *ZUM '98: The Z Formal Specification Notation: 11th International Conference of Z Users, Berlin, Germany, September 24-26, 1998. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 1998;367-406.

150. Woodcock J. Structuring specifications in Z. *Softw Eng J*. 1989;4(1):51-66.

151. Woodcock J. Mathematics as a management tool: proof rules for promotion. In: *Software Engineering for Large Software Systems*. Dordrecht, The Netherlands: Springer Netherlands; 1990;345-365.

152. Woodcock J, Stepney S, Cooper D, Clark J, Jacob J. The certification of the Mondex electronic purse to ITSEC Level E6. *Form Asp Comput*. 2008;20(1):5-19.

153. Bowen JP. Z: a formal specification notation. In: Frappier M, Habrias H, eds. *Software Specification Methods: An Overview Using a Case Study*. London, UK: Springer; 2001;3-19.

154. Freitas L, Woodcock J. Mechanising Mondex with Z/Eves. *Form Asp Comput*. 2008;20(1):117-139.

155. Smith G, Derrick J. Specification, refinement and verification of concurrent systems – an integration of Object-Z and CSP. *Formal Methods Syst Des*. 2001;18(3):249-284.

156. Cavalcanti A, Sampaio A, Woodcock J. A refinement strategy for *Circus*. *Form Asp Comput*. 2003;15(2):146-181.

157. Baumann P, Lermer K. Specifying parallel and distributed real-time systems in Z. In: Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS); 1996; Honolulu, HI.

158. Mirian-HosseinAbadi S-H, Mousavi MR. Making nondeterminism explicit in Z. In: Proceedings of the Iranian Computer Society Annual Conference (CSICC); 2002; Tehran, Iran.

159. Plagge D, Leuschel M. Validating Z specifications using the ProB animator and model checker. In: Davies J, Gibbons J, eds. *Integrated Formal Methods: 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007. Proceedings*. Berlin, Germany: Springer Berlin Heidelberg; 2007:480-500. *Lecture Notes in Computer Science*; vol 4591.

160. Spivey JM. Specifying a real-time kernel. *IEEE Softw*. 1990;7(5):21-28.

161. Malik P, Utting M. CZT: a framework for Z tools. In: *ZB 2005: Formal Specification and Development in Z and B: 4th International Conference of B and Z Users, Guildford, UK, April 13-15, 2005. Proceedings*. Berlin, Germany: Springer; 2005;65-84.

How to cite this article: Mashkoor A, Kossak F, Egyed A. Evaluating the suitability of state-based formal methods for industrial deployment. *Softw Pract Exper*. 2018;48:2350–2379. https://doi.org/10.1002/spe.2634