How to Select the Suitable Formal Method for an Industrial Application: A Survey

Felix Kossak and Atif Mashkoor(✉)

Software Competence Center Hagenberg GmbH,
Hagenberg, Austria
{felix.kossak,atif.mashkoor}@scch.at

Abstract. The share of formal methods is still marginal in contemporary systems and software engineering. One of the reasons is the absence of systematic guidelines and evaluation criteria that help software practitioners choose the right formal method for the problem at hand. In this paper, we present a comprehensive set of criteria, based on a systematic literature review and decade-long personal experience in industrial projects, for evaluating and comparing different formal methods. We argue that besides technical grounds (e.g., modeling capabilities and supported development phases), formal methods should also be evaluated from social and industrial perspectives. At the end of the paper, we present an evaluation of "ABZ" methods based on the stipulated criteria.

1 Introduction

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

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

The research presented in this paper is 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. The writing of the paper is partially supported by the Austrian Science Fund project: *Behavioral Theory and Logics for Distributed Adaptive Systems* (FWF-P26452-N15).

M. Butler et al. (Eds.): ABZ 2016, LNCS 9675, pp. 213–228, 2016.
DOI: 10.1007/978-3-319-33600-8_13

or whether the methods used must be suitable for ordinary software developers. Several studies have already been published where individual formal methods are compared. However, as we will detail in Sect. 2, many of these studies are either outdated or concentrate on limited aspects (e.g., technical criteria of predominantly academic interest or a particular domain of application). None has presented general guidelines/evaluation criteria which may help software practitioners in choosing the right formal method for their problem at hand.

In this paper, we present a comprehensive list of criteria for the comparison of formal methods with respect to general industrial interest, drawn from a structured literature review as well as personal experience in industrial and academic projects, for example, hemodialysis machines [42], an aircraft landing gear system [34], machine control systems [43], transportation systems [45], platooning systems [44], and business process modeling [35]. In contrast to many other publications, we include a wide range of criteria which we deem crucial for a wider adoption of formal methods in the industry.

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

The prime research question of our work is: What criteria are useful in order to select a particular formal method for a particular setting? Additionally, we demonstrate the use of the criteria with several selected formal methods in tabular form. A much more detailed description of the criteria and our evaluation of different methods is available in [36].

This paper is structured as follows: First we present our research approach and the list of literature reviewed (Sect. 2). Then in Sect. 3, we present a structured list of criteria for selecting a suitable formal method for an industrial application. In Sect. 4, we compare particular methods by means of the previously described criteria. The paper is concluded in Sect. 5.

2 Approach and Literature Reviewed

2.1 The Research Approach

In this paper, we answer the following research questions:

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

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

– “formal methods” AND “evaluation criteria”
– “formal methods” AND “comparison”
– “formal methods” AND “state of the art”
– “formal methods” AND “literature review”

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

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

Evaluations of formal methods from the 1990s must certainly be considered outdated, for much has changed since, in particular with respect to tool support, the amount of practical experience, and how widespread a method is used. This does not leave much material for a concrete evaluation of particular methods. Still, older publications can yield interesting contributions to the *criteria* by which formal methods should be evaluated (sometimes presented as wishes). Often the focus is on a purely academic viewpoint in this respect, but not always.

2.2 Literature Reviewed

We now present the literature (in order of relevance) which we found relevant for the current study (excluding sources on a single method).

Information on concrete evaluations within the industry appears to be scarce, though we assume that such evaluations happen. A notable exception is a recent paper by Chris Newcombe, “Why Amazon Chose TLA+” [50], though it largely only describes experience with TLA+ [38] and, to a lesser extent, Alloy [29] and Microsoft VCC [17]. The criteria are drawn from the very demanding domain of cloud infrastructure services, where key demands include a high level of distribution, high performance, and high availability.

A position paper by Sifakis [54] also discusses industry-centric evaluation criteria and provided useful input for us. Sifakis discusses, amongst others, the crucial point of usability and human factors in general.

The papers from Ardis et al. [4] and Knight et al. [33] also provide frameworks for the evaluation of formal specification languages. They first present criteria and then evaluate several formal languages. The latter also present the perspectives of developers, engineers, and computer scientists on these languages.

Woodcock et al. [58] contribute an overview of historical experiences with formal methods, in particular from industrial projects. We could extract several important criteria from this paper, in particular with regard to tool support.

McGibbon [46] discusses different evaluation criteria from a government viewpoint, including more detailed requirements for tools.

Several evaluation criteria can be extracted from the seminal papers by Clarke and Wing [15] and Bowen and Hinchey [13]. The former present the state of the art and future directions of formal methods and the latter present some guidelines to help propagate the use of formal methods in industry.

Liu et al. [40] list a number of evaluation criteria and compare a great number of methods. Amongst others, the authors bring in the additional criterion of applicability in re-engineering, in particular in reverse engineering and restructuring. Although they are primarily concerned with support for re-engineering, this paper is also of general interest; it includes interesting characterizations of many different methods and their state at the time, though unfortunately much of this information is now (potentially) outdated.

Banach, in "Model Based Refinement and the Tools of Tomorrow" [5], compares B [2], Event-B [3], Z [55], and the ASM method [10] from a mathematical/technical point of view.

Also a book by Gannon, Zelkowitz, and Purtilo, entitled *Software Specification: A Comparison of Formal Methods* [23], focuses on mathematical issues; it discusses only VDM [30] as a formal method in a closer sense, together with temporal logic in general as well as "risk assessment."

Also Kaur et al. in "Analysis of Three Formal Methods - Z, B and VDM" [32], stress mathematical and modeling issues, but they also mention, e.g., tool support, code generation, and testing.

In *Software Specification Methods* (ed. by Frappier and Habrias) [21], many different methods are introduced through a case study. In the last chapter, some of the methods are qualitatively compared. The criteria include some which we chose not to adopt here, including graphical representation (lack of relevance for state-based methods), object-orientated concepts (design-centric, see further below), use of variables (too detailed), and event inhibition (too detailed).

In "A practical comparison of Alloy and Spin," Zave [59] compares Alloy and Spin/Promela [28], two methods of general interest. However, we did not find any new criteria there.

In a master's thesis, Rainer-Harbach [53] compares several different proving tools for software verification, but not any comprehensive method which could support other project phases and aspects.

ter Beek et al. [9] wrote a paper on "Formal Methods for Service Composition," dealing with a very narrow field of application. They compare only automata as a basis for model checking, Petri Nets, and process algebras.

A paper by Dondossola [18] specializes on the application domain of safety-critical knowledge-based components and on the method TRIO [25]. Towards the end, it also offers a comparison of different formal methods, including VDM and Z; however, the criteria used there are only very coarsely described, so we could not extract much extra information useful for our purposes.

A technical report by Barjaktarovic [8] names in particular industrial requirements for formal methods throughout the text; most of those requirements are also found in other sources, but this paper provides a good confirmation.

From an article by Pandey and Batra [52], we obtained useful assessments of Z and VDM, in particular.

3 Criteria for Evaluating Formal Methods

Now we will present a structured list of criteria which we deem relevant for assessing and comparing formal methods for their usefulness in concrete industrial projects, depending on the concrete settings of a project. We first give an overview of the criteria we found and deemed relevant before describing each of them in more detail and explaining their significance. Please note that the classification of certain criteria under a particular category may be cross-cutting and overlapping to some degree. This is by choice as it makes each category an independent unit of analysis that can also be taken into consideration in isolation for concentration on a particular class of criteria. Please see [36] for a detailed discussion on the criteria.

3.1 Overview

We found five categories of criteria relevant for industrial projects:

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

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

3.2 Modeling Criteria

Modeling criteria concern the scope of systems and requirements which can be modeled. *Composition* is important for large models, including their verification. *Refinement* is even more important for constructing increasingly large models and can support validation, design, and coding.

Support for modeling *parallelism, concurrency, and distribution* is essential for a wide range of real-life applications. *Support for non-determinism* is very

useful for keeping models abstract. *The possibility to express global properties of system correctness* is necessary to be able to prove respective safety and liveness requirements such as temporal constraints (termination, deadlock freeness, fairness). *Support for modeling time* must regard sparse and dense models for time separately (see [40]). *Performance properties* refer to the complexity of algorithms, both with respect to time and to memory. Many domains of application require that *special concepts* be easily expressed in a modeling language. One important example is hybrid systems.

We can generally expect a desire in industry to "be able to capture rich concepts without tedious workarounds" [50]. In a related note, [15] demand support for sufficient data structures and algorithms.

Additionally, [21,32] 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.

3.3 Supported Development Phases

[15] state that it should be possible to amortize the cost of a formal method or tool over many uses; this means it should be possible to use a model throughout as many development phases as possible.

A special phase which is not regularly considered 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 [40].

Bug diagnosis is an issue which deserves special mention beside verification, because finding that some property does not hold does not mean that one can then easily identify the source of error. [50] points out the importance of this issue; [8] states even that "Industry is mostly interested in tools that find bugs rather than tools that prove correctness."

3.4 Technical Criteria

Technical criteria concern tool support and how the method and the available tools interact with other aspects of system development. Besides the range of *overall tool support*, the availability of professional support for those tools is important, for which reason companies typically prefer *commercial support*.

An important issue stressed by many industrial sources is the *traceability of requirements* throughout the development process. *Support for change management* addresses the fact that the waterfall model is actually unrealistic. The *effect of the method on overall development time* is crucial for the industry.

Regarding code generation, we can consider the efficiency of the generated code as well as the efficiency of code generation. The *efficiency of the generated code* is the quality of the code that has been generated by an automatic tool from a more abstract model: runtime behavior, use of memory, or the amount

of manual fixing which is required after generation. The *efficiency of code generation* concerns the speed (and use of resources) with which code is generated. This is important for "playing" with the model and testing different designs.

The demand for *interoperability with other methods and/or other tools* arises from the insight that different methods and tools are differently suitable for different tasks and project phases. Moreover, such a possibility will greatly enhance reuse. A related criterion is *Integration of methodology and tools with the usual development methods and tools* to facilitate the transition between different project phases and requirements tracking, amongst others.

3.5 Human/Social Criteria

In industrial settings, specially trained people will not be available for every development task. 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, managers, or even lawyers (cf. [37]).

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. *General understandability* is important because formal models often need to be understood by various stakeholders. The importance of *documentation*, including reference handbooks or tutorials, is self-evident. *Support for collaboration* is easily forgotten when academics develop a new method, but it is an important issue in larger real-life projects.

3.6 Industrial Applicability

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.

The criterion of *support for industrial deployment* is designed to capture the availability of outside help. *Scalability* is the ability of the method to be well applicable to arbitrarily large and complex projects. Certainly the actual *amount of industrial experience* which has been gathered with a method is very interesting for decision makers who ponder newly introducing formal methods. Also the *success rate* would be interesting, but would be extremely difficult to assess objectively. A cliché that formal methods would require *specially trained, "expensive" personnel* is actually well-founded. There are considerable differences between particular methods in this respect.

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.

Related is the availability and *licensing* of the method and related tools. Most of the widely used methods and their tools are open source, but open-source

software requires a large and stable community to maintain and further develop. Moreover, the availability of commercial support and training is essential for more widespread uptake in the industry.

4 Comparison of Methods

4.1 Comparison

We now compare the different "ABZ" methods through simplified tables, see Tables 1, 2, 3, 4 and 5.

Table 1. Modeling criteria

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

Table 2. Supported development phases

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

In the tables,"Y" means "yes/supported" (quality unknown), "N" means "not supported." A dash "-" means that we could not find (sufficient) information. A "?" means that we have inconsistent or even contradicting information. "(Y)" means restricted support, "(N)" means little support, "(Good)" means "Good" with some proviso, etc.; parentheses may also indicate that special versions or prototypes support this feature, but not the standard version.

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

Table 4. Human/social criteria

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

Table 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	(Good)
Experience	(Much)	Med	Much	(Much)	Much	Much	Much
Special staff	Y	(N)	Y	Y	N	(N)	Y
Standardization	N	N	N	N	N	Y	Y
Licensing	OS	OS	Cm	OS	OS	Cm/OS	OS

"Med." abbreviates medium quality, "Part." partial support, "Man." manual, i.e., no tool support, "Impl." abbreviates only implicit support, and "Adapt." adaptable. "Cm." abbreviates "commercial" (licensing), "OS" open source. "n/a" means "not applicable." The entry "Hybr." denotes the possibility to model hybrid (discrete-continuous) systems.

A couple of criteria have been omitted due to either uniform support or lack of information: in Table 2, *specification*, *validation* and *performance checking*; and in Table 3, *change management*.

4.2 Justification

A detailed justification of the entries in the above tables is given in [36]. Here we only give an extract of the potentially contentious points regarding the modeling criteria.

Alloy features an explicit *composition* operation [22]. Temporal composition of functions is possible with operators `merge` and `override`. According to [50], *refinement* is not very flexible. Regarding *concurrency*, according to [50], the method is not suitable for large complex systems. *Non-determinism* can only be implicitly modeled [59]. For examples of the modeling of *global system properties*, see e.g., [14,16,31]. Alloy has no direct *notion of time* [24]; however, [1,19,57] have shown how to express timing properties. [41] selected Alloy for its "expressive power," amongst others, but according to [50], the *expression of rich concepts* is *not* easy.

For **ASMs**, *(de-)composition* is judged as medium by [5]; however, from a practical point of view, we consider it to be quite flexible. *Refinement* is good, according to [5] as well as by our own experience. The ASM method supports n-to-m refinement and both data refinement and procedural refinement [10]. ASMs are well suited for modeling *parallel and concurrent systems* [20]. *Non-determinism* is supported by the "choose" operator and via abstract rules and derived functions. *Global properties* can be expressed via the state space, but there is no explicit support. There is no explicit *notion of time*. Regarding *special concepts*, Banach and others have used ASMs for modeling continuous systems [7]. Regarding *easy expression of rich concepts*, the simple notation can be easily adapted and expanded, but tool support may always be limited.

In **B**, *(de-)composition* is possible by including other machines. According to Banach [5], "The [...] INCLUDES, USES, SEES mechanisms are certainly composition mechanisms, but they just act at the top level." B supports only 1-to-1 *refinement* (cf. [5]), and also as per our own experience, the support for refinement in B can be rated as 'medium." B supports *parallelism*, except for code generation, but not *concurrency* [32,40]. B supports *non-determinism* through non-deterministic choice of values as well as by operators "`ANY`" and "`CHOICE`". Regarding *system properties*, it is possible to express typical safety properties through invariants, but B has no explicit means for modeling *time* or *temporal properties* (see also [40]). *Reliability* properties can be expressed via invariants. Regarding the *easy expression of rich concepts*, B provides a rich language for set theory and relational theory. However, expressing certain concepts such as data structures can often be awkward and unintuitive.

For **Event-B**, [5] assesses *(de-)composition* as "good"; however, we find the decomposition/recomposition facilities not straightforward. Event-B only supports 1-to-1 *refinement*. Event-B does not explicitly support *parallelism and concurrency*; however, both parallel (cf. [27]) and concurrent (cf. [11]) programs can be defined using decomposition and refinement. Event-B supports *non-determinism* by allowing for non-deterministic choice of values for variables and through event parameters. *Global system properties* can effectively be specified using invariants. Event-B has no explicit means for *modeling time or temporal*

properties. Regarding *special concepts*, there exist proposals regarding hybrid and continuous systems, e.g., in [6]. Regarding the *easy expression of rich concepts*, our comments on B apply here as well.

TLA+ supports *composition* through different mechanisms such as logical connectives of implication, conjunction, and quantification [38,47]. *Refinement* is assessed as "good" by [50, p. 28]; according to [47, p. 445], "A distinctive feature of TLA is its attention to refinement and composition." Support for modeling *parallel, concurrent, and distributed systems* is good, as confirmed by [50, p. 36]. *Non-determinism* is also supported [38, Section 6.6]. TLA+ does not formally distinguish between specifications and *system properties*: both are written as logical formulas and concepts such as refinement and composition [47]. It uses set theocratic constructs to define safety properties and temporal logic to define liveness properties. *Modeling of time* is explicitly supported [51, p. 69], enabling modeling and checking of *performance properties*. According to [50, pp. 27, 36], *rich concepts can be easily expressed* in TLA+.

In **VDM**, *Composition* is possible according to [40], but [32,46] deny it. Classical VDM models can be structured into data types and modules, while VDM++ models can be structured into classes. *Refinement* is achieved through data reification and operation decomposition. According to [40,46], VDM does not support *parallelism*, but [39] describes the use of VDM for *distributed*, embedded, *real-time* systems. [52] state that "VDM emphasizes on the feature of concurrency control" (cf. [56]). Support for *non-determinism* is only given in VDM++ [32]. [46] states that VDM has no explicit *notion of time*, but [39] describes timing analysis for identifying performance bottlenecks. [49,56] also deal with real-time systems. Regarding *special modeling concepts*, [46,52] note that VDM has explicit *exception handling*. Support for e.g., performance and reliability modeling has been introduced more recently.

In **Z**, *(De-)Composition* is achieved through "schemas" [5,12] or by means of "promotion" [5]. [5] notes that the schema calculus is not monotonic with respect to refinement. Also taking [32] into account, we assess composition as medium. Regarding *refinement*, [5] notes that "spurious traces, not corresponding to real world behaviour, can be generated." Refinement cannot completely go down to the code level [12]. Z does not directly support *concurrency* [40,46,52]. Regarding *non-determinism*, [12,21] state it is given but [32] denies it. Z does not support non-determinism explicitly, but e.g., several after-state valuations for a single pre-state binding are possible (cf. [48]). *Expression of global system properties* is not straightforward (cf. [26]). An explicit *concept of time* is obviously not given [40,46].

4.3 Project-Specific Assessment

We have furthermore tried to condense the available information into a much simplified table that can be found in [36] for fast management decisions. Here, we just sketch the structure of the table as follows:

- **Project setting:** Is the product safety-critical? How severe is time pressure? Is the project conducted in an agile setting? Does the method allow to quickly

start using it without prior experience – at least with *initial* help by experts? Or will continued support by experts be required?
- **Company:** Do we deal with a big company or with a small or medium-sized company? Can the company afford a transition phase for introducing formal methods?
- **Goal of using formal methods:** Do we want to improve product quality, or process quality? Reduce specification errors? Improve requirements definitions, documentation, understanding of the design? Explore a model before implementation? Obtain a sound foundation for maintenance and/or testing? Meet safety requirements?

5 Conclusion

The main contribution 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.

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

We hope that our work will contribute to better acceptance of formal 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.

References

1. Abdunabi, R., Sun, W., Ray, I.: Enforcing spatio-temporal access control in mobile applications. Computing **96**(4), 313–353 (2014)
2. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
3. Abrial, J.R.: Modeling in Event-B System and Software Design. Cambridge University Press, Cambridge (2010)
4. Ardis, M.A., Chaves, J.A., Jagadeesan, L.J., Mataga, P., Puchol, C., Staskauskas, M.G., Von Olnhausen, J.: A framework for evaluating specification methods for reactive systems. IEEE Trans. Softw. Eng. **22**(6), 378–389 (1996)
5. Banach, R.: Model based refinement and the tools of tomorrow. In: Börger, E., Butler, M., Bowen, J.P., Boca, P. (eds.) ABZ 2008. LNCS, vol. 5238, pp. 42–56. Springer, Heidelberg (2008)
6. Banach, R., Zhu, H., Su, W., Huang, R.: Formalising the continuous/discrete modeling step. In: Proceedings Refine 2011. EPTCS, vol. 55, pp. 121–138 (2011)
7. Banach, R., Zhu, H., Su, W., Wu, X.: A continuous ASM modelling approach to pacemaker sensing. ACM Trans. Softw. Eng. Methodol. **24**(1), 2 (2014)

8. Barjaktarovic, M.: The state-of-the-art in formal methods. Technical report/Wilkes University and WetStone Technologies (1998). http://www.cs.utexas.edu/users/csed/formal-methods/docs/StateFM.pdf
9. ter Beek, M.H., Bucchiarone, A., Gnesi, S.: Formal methods for service composition. Ann. Math. Comput. Teleinformatics **1**(5), 1–10 (2007)
10. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, Berlin (2003)
11. Boström, P., Degerlund, F., Sere, K., Waldén, M.: Derivation of concurrent programs by stepwise scheduling of Event-B models. Formal Aspects Comput. **26**(2), 281–303 (2014)
12. Bowen, J.P.: Z: a formal specification notation. In: Frappier, M., Habrias, H. (eds.) Software Specification Methods: An Overview Using a Case Study, pp. 3–19. Springer, London (2001)
13. Bowen, J.P., Hinchey, M.G.: Ten commandments of formal methods. Computer **28**(4), 56–63 (1995)
14. Brunel, J., Rioux, L., Paul, S., Faucogney, A., Vallée, F.: Formal safety and security assessment of an avionic architecture with Alloy. In: Third International Workshop on Engineering Safety and Security Systems (ESSS 2014), pp. 8–19 (2014)
15. Clarke, E.M., Wing, J.M.: Formal methods: state of the art and future directions. ACM Comput. Surv. **28**(4), 626–643 (1996)
16. Cochran, D., Kiniry, J.R.: Formal model-based validation for tally systems. In: Heather, J., Schneider, S., Teague, V. (eds.) Vote-ID 2013. LNCS, vol. 7985, pp. 41–60. Springer, Heidelberg (2013)
17. Cohen, E., Dahlweid, M., Hillebrand, M., Leinenbach, D., Moskal, M., Santen, T., Schulte, W., Tobies, S.: VCC: a practical system for verifying concurrent C. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 23–42. Springer, Heidelberg (2009)
18. Dondossola, G.: Formal methods in the development of safety critical knowledge-based components. In: Proceedings of the KR 1998 European Workshop on Validation and Verification of Knowledge-Based Systems, pp. 232–237 (1998)
19. Dwivedi, A.K., Rath, S.K.: Model to specify real time system using Z and Alloy languages: a comparative approach. In: International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012), pp. 1–6 (2012)
20. Ferrarotti, F., Schewe, K., Tec, L., Wang, Q.: A new thesis concerning synchronised parallel computing - simplified parallel ASM thesis. CoRR abs/1504.06203 (2015)
21. Frappier, M., Habrias, H. (eds.): Software Specification Methods. ISTE, London (2006)
22. Frias, M.F., Pombo, C.G.L., Aguirre, N.M.: An equational calculus for alloy. In: Davies, J., Schulte, W., Barnett, M. (eds.) ICFEM 2004. LNCS, vol. 3308, pp. 162–175. Springer, Heidelberg (2004)
23. Gannon, J.D., Zelkowitz, M.V., Purtilo, J.M.: Software Specification: A Comparison of Formal Methods. Greenwood Publishing, Westpoint (1994)
24. Georg, G., Bieman, J., France, R.B.: Using Alloy and UML/OCL to specify run-time configuration management: a case study. In: Workshop of the pUML-Group Held Together with the UML 2001 on Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists, pp. 128–141, GI (2001)
25. Ghezzi, C., Mandrioli, D., Morzenti, A.: TRIO: a logic language for executable specifications of real-time systems. J. Syst. Softw. **12**(2), 107–123 (1990)
26. Haughton, H.P.: Using Z to model and analyse safety and liveness properties of communication protocols. Inf. Softw. Technol. **33**(8), 575–580 (1991)

27. Hoang, T.S., Abrial, J.-R.: Event-B decomposition for parallel programs. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) ABZ 2010. LNCS, vol. 5977, pp. 319–333. Springer, Heidelberg (2010)
28. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley, Reading (2004)
29. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press, Cambridge (2006)
30. Jones, C.B.: Systematic Software Development Using VDM, 2nd edn. Prentice-Hall Inc., Upper Saddle River (1990)
31. Kang, E., Jackson, D.: Formal modeling and analysis of a flash filesystem in Alloy. In: Börger, E., Butler, M., Bowen, J.P., Boca, P. (eds.) ABZ 2008. LNCS, vol. 5238, pp. 294–308. Springer, Heidelberg (2008)
32. Kaur, A., Gulati, S., Singh, S.: Analysis of three formal methods - Z, B and VDM. Int. J. Eng. Res. Technol. (IJERT) **1**(4), 1–4 (2012)
33. Knight, J.C., DeJong, C.L., Gibble, M.S., Nakano, L.G.: Why are formal methods not used more widely? In: The Fourth NASA Langley Formal Methods Workshop (LFM 1997) (1997)
34. Kossak, F.: Landing gear system: an ASM-based solution for the ABZ case study. In: Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, K.-D. (eds.) ABZ 2014. CCIS, vol. 433, pp. 142–147. Springer, Heidelberg (2014)
35. Kossak, F., Illibauer, C., Geist, V., Kubovy, J., Natschläger, C., Ziebermayr, T., Kopetzky, T., Freudenthaler, B., Schewe, K.D.: A Rigorous Semantics for BPMN 2.0 Process Diagrams. Springer, Heidelberg (2015)
36. Kossak, F., Mashkoor, A.: How to Evaluate the Suitability of a Formal Method for Industrial Deployment? A Survey. Technical report SCCH-TR-1603, Software Competence Center Hagenberg GmbH, Hagenberg, Austria (2016). http://www.scch.at/en/rse-news/fm_comparison
37. Kossak, F., Mashkoor, A., Geist, V., Illibauer, C.: Improving the understandability of formal specifications: an experience report. In: Salinesi, C., van de Weerd, I. (eds.) REFSQ 2014. LNCS, vol. 8396, pp. 184–199. Springer, Heidelberg (2014)
38. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, Boston (2002)
39. Larsen, P.G., Wolff, S.: Development process of distributed embedded systems using VDM, Overture – Open-source Tools for Formal Modelling TR-2010-02 (2010)
40. Liu, X., Yand, H., Zedan, H.: Formal methods for the re-engineering of computing systems. In: Proceedings of the 21st Computer Software and Applications Conference (COMPSAC 1997), pp. 409–414 (1997)
41. Maoz, S., Ringert, J.O., Rumpe, B.: Semantically configurable consistency analysis for class and object diagrams. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 153–167. Springer, Heidelberg (2011)
42. Mashkoor, A., Biro, M.: Towards the trustworthy development of active medical devices: a hemodialysis case study. IEEE Embed. Syst. Lett. **8**(1), 14–17 (2016)
43. Mashkoor, A., Hasan, O., Beer, W.: Using probabilistic analysis for the certification of machine control systems. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CD-ARES Workshops 2013. LNCS, vol. 8128, pp. 305–320. Springer, Heidelberg (2013)
44. Mashkoor, A., Jacquot, J.P.: Stepwise validation of formal specifications. In: 18th Asia-Pacific Software Engineering Conference (APSEC 2011), pp. 57–64. IEEE, Ho Chi Minh City, Vietnam (2011)

45. Mashkoor, A., Jacquot, J.P.: Utilizing Event-B for domain engineering: a critical analysis. Requirements Eng. **16**(3), 191–207 (2011)
46. McGibbon, T.: An analysis of two formal methods: VDM and Z. Technical report, DoD Data and Analysis Center for Software (DACS) (1997). https://www.csiac.org/sites/default/files/An%20Analysis%20of%20Two%20Formal%20Methods%20-%20VDM%20and%20Z%20-%20SOAR.pdf
47. Merz, S.: The specification language TLA+. In: Bjørner, D., Henson, M. (eds.) Logics of Specification Languages. Monographs in Theoretical Computer Science, pp. 401–451. Springer, Heidelberg (2008)
48. Mirian-HosseinAbadi, S.H., Mousavi, M.R.: Making nondeterminism explicit in Z. In: Proceedings of the Iranian Computer Society Annual Conference (CSICC 2002), Tehran, Iran (2002)
49. Mukherjee, P., Bousquet, F., Delabre, J., Paynter, S., Larsen, P.G.: Exploring timing properties using VDM++ on an industrial application. In: Proceedings of the Second VDM Workshop (2000)
50. Newcombe, C.: Why Amazon chose TLA$^+$. In: Ait Ameur, Y., Schewe, K.-D. (eds.) ABZ 2014. LNCS, vol. 8477, pp. 25–39. Springer, Heidelberg (2014)
51. Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., Deardeuff, M.: How amazon web services uses formal methods. Commun. ACM **58**(4), 66–73 (2015)
52. Pandey, S., Batra, M.: Formal methods in requirements phase of SDLC. Int. J. Comput. Appl. **70**(13), 7–14 (2013)
53. Rainer-Harbach, M.: Methods and tools for the formal verification of software. An analysis and comparison. Diplomarbeit, Fakultät für Informatik, Technische Universität Wien. https://www.ads.tuwien.ac.at/publications/bib/pdf/rainer-harbach_11.pdf
54. Sifakis, J.: Formal methods and their evaluation. Position Paper Presented at FEMSYS in Munich (1997). http://www-verimag.imag.fr/~sifakis/RECH/FEMSYS/paper.ps
55. Spivey, J.M.: The Z Notation: A Reference Manual. Prentice-Hall Inc., Upper Saddle River (1989)
56. Verhoef, M., Larsen, P.G., Hooman, J.: Modeling and validating distributed embedded real-time systems with VDM++. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 147–162. Springer, Heidelberg (2006)
57. Wang, T., Ji, D.: Active attacking multicast key management protocol using Alloy. In: Derrick, J., Fitzgerald, J., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) ABZ 2012. LNCS, vol. 7316, pp. 164–177. Springer, Heidelberg (2012)
58. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.: Formal methods: practice and experience. ACM Comput. Surv. **41**(4), 19 (2009)
59. Zave, P.: A practical comparison of Alloy and Spin. Formal Aspects Comput. **2015**(2), 239–253 (2015)