Investigating the Usefulness of Formal Methods for Developing Industrial Software Systems: A Systematic Literature Review

Alex Pawelczyk
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

Abstract

The objective of this paper is to present the current evidence relative to the usefulness of formal methods (FMs) for developing industrial software systems. As such insight is important for project managers of software development firms, this work presents a systematic literature review (SLR) of empirical studies that investigate if applying FMs and tools to the software development process of industrial computing applications can be useful. The SLR synthesizes evidence from 12 primary studies and analyzes how useful FMs are when applied by professionals in industry. The results of the SLR suggest that FMs are useful in a wide range of domains when applied to the development process of safety-critical systems. Using FMs and tools can lead to a lower defect rate, less ambiguity in system specifications, and a shorter time to market. Non-experts benefit from user-friendly FM tools, but the main drawback of FMs is the initial learning time required to learn a method or tool.

Keywords: formal methods, systematic literature review

ACM Reference Format:
Alex Pawelczyk. 2020. Investigating the Usefulness of Formal Methods for Developing Industrial Software Systems: A Systematic Literature Review. In *CS646: Software Design and Architecture, Aug. 2, 2020, Waterloo, ON*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

1 Introduction

Modern-day software systems are becoming increasingly dependant on software components, the complexity of systems with embedded software is rapidly growing, and maintaining reliability in software-intensive systems is difficult to do [30]. A popular area of research that can help mitigate these challenges is the application of formal methods (FMs) to the development process of industrial software systems. FMs are mathematically rigorous techniques and tools that are used for the specification, design, and verification of software and hardware systems [13]. FMs aim to improve the reliability and dependability of software systems by supporting program development and providing a foundation for describing and reasoning about complex systems. In addition, FMs can reveal ambiguity, incompleteness, and inconsistency in a system [61].

A problem with FMs is that their usefulness in the development of industrial software applications is one of the most controversial issues and source of debates within the software engineering community [10]. Many people argue that the use of FMs during the software development produces better quality programs of the highest integrity, reduces the time needed for software development, and decreases the cost of development [9, 10, 22, 53, 65]. Others argue that FMs drive up the initial cost of product development, increase time to market, do not scale well with the size and complexity of real-world systems, and cannot be applied to certain classes of systems (e.g., those that employ machine learning) [37, 46, 47, 51]. Thus, the primary motivation of this work is to review the state of the art usage of FMs in industrial domains and identify the successes and failures.

A secondary motivation of this research stems from the fact that there have been many reports on successful applications of FMs to real-world industrial projects, but the widespread adoption of FMs in industrial domains is still an exception [4, 33, 58]. Project managers at software development firms may be hesitant to adopt new practices unless there is reliable evidence that points to the success of a particular technique. Since the results of one study are not enough to draw meaningful conclusions about the general usefulness of FMs in industrial settings, it is important to conduct research that summarizes the existing evidence concerning this topic. Moreover, synthesizing evidence from a large number of studies may lead to the discovery of open research questions that need to be addressed before FMs become widely adopted in industry.

CS646: Software Design and Architecture, Aug. 2, 2020, Waterloo, ON

https://doi.org/10.1145/1122445.1122456

This work presents a systematic literature review (SLR) that investigates the usefulness of FMs and tools when applied to the software development process of industrial software systems. The primary objective of the SLR is to summarize all existing information about the state of the art usage of FMs in industrial domains in a thorough and unbiased manner. The SLR derives conclusions from the results of 12 primary studies into a scientific summary of evidence relevant to the degree of usefulness that FMs have for developing industrial applications. Additionally, the SLR presents any conflicting findings from the analysis, identifies gaps in the existing body of knowledge, discusses the implications of these results for managers and executives of software development companies, and suggests areas for future research.

The rest of this paper is organized as follows. Section 2 summarizes the results of past SLRs and surveys on FMs. Section 3 backgrounds two key branches of FMs: *formal specification* and *formal verification*. Section 4 provides an overview of the review protocol that is applied to the SLR. Section 5 details how the SLR is conducted. Section 6 reports the results of the SLR based on the synthesis of evidence from a collection of empirical studies. Section 7 discusses the key findings and implications of the SLR. Section 8 identifies the limitations of the SLR. Section 9 proposes areas for future work in FMs research, and Section 10 presents concluding remarks.

2 Related Work

In 1996, Clarke et. al [15] assessed the state of the art in formal specification and verification, focusing on model checking and theorem proving. The authors propose definitions for formal specification, model checking, and theorem proving, along with case studies where FMs were used to successfully specify commercial and safety-critical software. Further case studies are presented where FMs were used to verify protocol standards and hardware designs. The authors also identify future directions for research in FMs, including fundamental concepts, methods and tools, integration of methods, and education and technology transfer. Concluding remarks state that the authors expect the role of FMs in the entire system-development process will increase, but progress will depend on continued support for research on new specification languages and new verification techniques.

In 1999, Kern and Greenstreet [26] surveyed a variety of formal techniques and frameworks that were proposed in the literature and incorporated into actual designs of hardware. The specification frameworks that they describe include temporal logic, predicate logic, abstraction, refinement, and the containment between w-regular languages. The verification techniques that they describe include model checking, automata-theoretic techniques, automated theorem proving, and hybrid approaches that integrate these verification techniques. Kern and Greenstreet also present a selection of case studies where FMs were applied to industrial-scale hardware designs, including microprocessors, floating-point hardware, protocols, memory subsystems, and communications hardware.

Woodcock et. al [65] described the current state of the art (as of 2009) in the industrial use of formal methods. The authors undertook a survey where data was collected between November 2007 and December 2008 on 62 industrial projects known to have employed FMs. The largest application domain where FMs were employed was transport, and other major domains were the financial sector, defence, and telecommunications. The effect on time taken for development was on average beneficial, where three times as many studies reported a reduction in time, rather than an increase. Several responses noted an increase in time in the specification phase, which may or may not have resulted in a decrease in time later in the development process. Five times as many projects reported reduced cost as opposed to an increase in cost, 92% of all cases reported an increase in software quality compared to other techniques, and no cases reported a decrease in quality. Improvements in software quality are related to the detection of faults (36%), improvements in design (12%), increased confidence in correctness (10%), improved understanding (10%), and early identification of faults or other issues (4%).

Weyns et. al [59] conducted an SLR based on 75 primary studies to identify trends in the application of FMs in self-adaptive systems, what kinds of FMs have been used in self-adaptive systems, and for which adaptation concerns were FMs used for. Although the authors found that there was an increase in the use of FMs in self-adaptive systems between 2001 and 2011, there was a lack of approaches that employ FMs for assuring that self-adaptive systems satisfy user requirements and meet the expected quality attributes. Weyns et. al also report that a majority of researchers employ regular algebraic notations for both modeling and property specification of self-adaptive systems. The top self-adaptation concerns that FMs are used for are efficiency and performance (32.0%), reliability (26.7%), guaranteeing functionality (22.7%), and flexibility (18.7%). Moreover, FMs were mostly used for the development of embedded systems (46.7%), followed by service-based systems (26.7%).

2019 saw an uptake in surveys on the applications of FMs to various domains. Nanda and Grant [41] focused on the use of formal specification in safety-critical systems and report that this method has made safety-critical software development more consistent, complete, and less ambiguous. They conclude that formal specification at the initial phase of software development is necessary and should be universally adopted for safety-critical systems. Sinha et. al [52] conducted a survey on static FMs for creating more dependable industrial automation systems. They conclude that static approaches are more useful during the earlier stages

of the software development life cycle, where bugs and inconsistencies are cheaper to find and correct. Murray and Anisi [39] surveyed formal verification techniques for smart contracts running on distributed ledgers such as blockchain. They conclude that the current state of the art formal verification methods are not suitable for complex contracts, but they can handle simple smart contracts and simplified models.

3 Two Key Branches of FMs

3.1 Formal Specification

A formal specification is the expression, in some formal or mathematical syntax and at some level of abstraction, of a collection of properties that some system should satisfy [32]. Formal specifications can be used to describe a system, analyze the behavior of a system, and verify key properties of interest through rigorous and effective reasoning tools [24]. The syntax that is used to write a formal specification is typically textual, but can also be graphical. A *semantics* is also provided, where a precise meaning is given to each description in the specification. This helps overcome many of the problems that arise from specifications written in a natural language, such as ambiguity.

One of the benefits of employing formal specifications in the software development process is that the quality of the software can be increased, leading to reduced debugging effort later on in the project cycle [24]. Developing a formal specification can also provide valuable insights and understanding of the software requirements and design. Moreover, formal specifications are crucial for designing, validating, documenting, communicating, reengineering, and reusing software solutions [32]. Formality helps obtain higher-quality specifications within such processes, while also providing the basis for their automated support.

3.2 Formal Verification

Formal verification uses the formal specifications of a hardware or software system to ensure that a design conforms to some precisely expressed notion of functional correctness [8]. As the complexity of a system design increases, the number of states to be exercised also increases. This results in a time-consuming task of generating and checking such states, along with an additional effort to assure acceptable test coverage metrics so the design can be considered fully verified and matching the specification. Formal verification enables the coverage of all possible states in a design by running a formal proof, rather than a simulation. Once the formal proof assures a statement as true, the negation of such statement is considered impossible to be reached [49].

Formal verification is beneficial because it can be used to conclusively prove that certain properties of a system always hold [8]. Formal verification can also lead to an increase in software quality and a reduction in time-to-market [63]. The underlying drivers that lead to a higher quality and a shorter time-to-market are early availability, higher coverage, and effective integration. Early availability is made possible because formal verification can start as soon as the logic of a design project is compiled, providing a limited setup cost relative to simulation testbenches. Formal verification can also cover large state spaces, leading to the detection of bugs which would have been difficult to manually target with simulation. In addition, using formal verification at the unit level can lead to a reduction in system simulation time and an increase in stabilization [63].

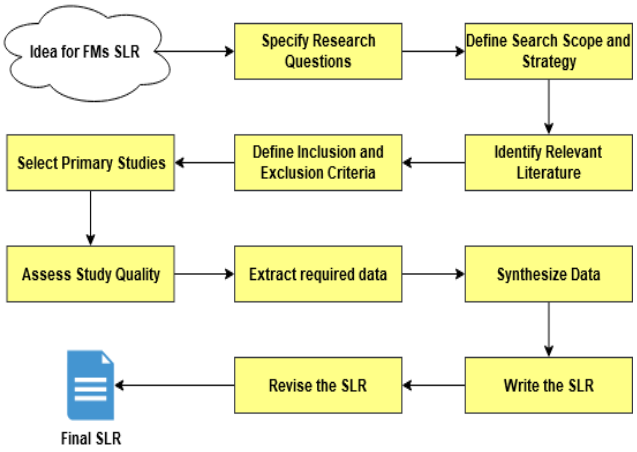

Figure 1. Overview of the SLR review protocol (adapted from [12, 59]).

4 The Review Protocol

This study follows the principles of an SLR, which is a method for identifying, evaluating, and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest [3]. One researcher (the author) was involved in conducting the SLR, and the first step involved defining the review protocol (see Figure 1).

The first step involves defining the research questions that guide this work. These questions form the foundation of the SLR, and answering these questions will provide interested stakeholders with valuable information about FMs in industry.

The next step of the review protocol focuses on identifying relevant literature that can help answer the research questions of interest. To help retain focus and organization, a search scope and strategy is defined before searching for relevant literature. During the initial search, any papers that may be relevant to the research questions are collected.

After collecting an initial pool of literature, primary studies are selected based on a set of inclusion and exclusion criteria. These criteria are used to ensure that the primary studies of the SLR provide relevant data towards answering the research questions.

Once the primary studies are identified, the quality of each study is manually assessed based on a set of quality criteria. Next, relevant data items (derived from the primary studies) are collated and summarized to present meaningful information. Using this information, evidence is synthesized in the form of a written report and presented in a manner that answers the research questions of this work.

5 Conducting the Review

5.1 Research Questions

The general goal of the SLR is formulated using the Goal Question Metric (GQM) approach [5]:

- *Purpose:* Understand and characterize
- *Issue:* the usefulness of FMs and tools
- *Object:* in the software development process of industrial software systems
- *Viewpoint:* from the perspectives of professional software developers and management.

From the specification of this general goal, meaningful research questions can be derived that characterize the goal in a quantifiable way:

- **RQ1:** "What types of FMs and tools are used for specifying and verifying industrial software systems?"
- **RQ2:** "How useful are FMs and tools when applied to industrial software systems?"

RQ1 is motivated by the need to gain insight into the types of FMs and tools that are being used in industry. FMs research has an extensive history, and there are a plethora of studies that propose novel FMs and tools that ease the integration process of FMs into the software development process. However, a key area of interest for this SLR is to find out which techniques are being used by real-world companies in industry. The aim is to identify if there is a group of FMs or tools that are being used more often in industry than others.

RQ2 focuses on identifying both successful and unsuccessful applications of FMs or tools in industrial settings. A method or tool is considered to be *useful* if it produces software of higher quality, increases knowledge about a system, shortens time to market, or lowers the project development cost compared to the normal business practices of an organization.

5.2 Identifying Relevant Literature

The scope of the search is defined by the dimensions of time and space. Regarding time, the SLR includes studies that were published between Jan. 2010 and Jun. 2020. 2010 was chosen as the starting year because the primary objective of the SLR is to summarize all existing information about the state of the art usage of FMs for developing industrial software systems. In the context of the SLR, state of the art refers to studies published within the last 10 years.

Regarding the space dimension, the SLR searches different online databases for publications related to FMs. The aim of an SLR is to find as many primary studies that relate to the research questions as possible using an unbiased search strategy [3]. Khan et. al [27] recommend searching multiple databases to obtain as many citations as possible and to avoid publication bias. Using the search string *("formal methods" OR "formal-methods") AND ("industry" OR "industrial")*, the first phase of identifying relevant literature involved searching the following three online databases: *ACM Digital Library*, *IEEE Explore*, and *ScienceDirect*.

After searching the online databases and downloading relevant studies, the identification of relevant literature continued into the second phase. Here, the *snowballing* technique was applied and the references of all papers found in the primary search phase were reviewed. Any references that were deemed suitable were added to the existing list of studies. The end of this phase resulted in a total of 192 primary study candidates.

5.3 Inclusion and Exclusion Criteria

After collecting a pool of relevant literature, primary studies were selected based on different inclusion and exclusion criteria. The primary inclusion criterion aims to select studies that investigate the usefulness of FMs for developing industrial software systems. These studies feature professional software developers using FMs during the development process of real-world projects. Moreover, the studies investigate how useful these techniques are. A secondary inclusion criterion targets studies that investigate the usefulness of FM tools in industry. Due to the complex nature of FMs, many companies resort to user-friendly tools that ease the integration of FMs into the software development process. Thus, investigating the usefulness of these tools is an important area of research.

The primary exclusion criterion is composed of studies that propose novel FMs and do not apply them to industrial projects. These studies tend to focus on the theory and details of the proposed methods, rather than demonstrating how they can be applied to real-word scenarios. Studies are also excluded if they meet at least one of the following criteria:

- ***Criterion 1***: FM studies that are targeted for CS/SE education.
  Rationale: These studies tend to focus on techniques that can help students effectively learn about formal methods, while the aim of this SLR is to analyze the usefulness of FMs when applied to the development process of industrial software systems.
- ***Criterion 2***: FM studies that are conducted in experimental settings.
  Rationale: Experiments typically use students as subjects, which are not a good representation of the professional programmers that apply FMs to industrial

projects. Even if an experiment uses professional developers as subjects, completing programming tasks during experiments typically takes a short amount of time, ranging anywhere from one hour to a few days. This is an unrealistic scenario in a real-world work environment, where projects are completed over a series of months, if not years. Moreover, it is difficult to match the complexity of an experimental programming task to that of an industrial-sized project.

- ***Criterion 3***: Studies written in languages other than English or Polish.
  Rationale: The author is only capable of reading in English or Polish. However, this may result in relevant literature being omitted from the SLR (see Section 8).

5.4 Quality Assessment

After the selection of 12 primary studies from the pool of 192 candidate studies, the quality of each primary study was assessed based on the 11 criteria used in the Dybå and Dingsøyr SLR of Agile methods (see Table 1) [18]. These criteria assess the reporting, rigor, credibility, and relevance of the remaining studies. Reporting is considered to be of high quality if the rationale, aims, and context of a study are clearly stated. Rigor refers to whether a thorough and appropriate approach was applied to the key research methods in a study. Studies are credible if the findings are well-presented and meaningful, and relevance describes the usefulness of findings to the software engineering research community and industry.

With regards to Table 1, questions 1-3 relate to the quality of the reporting of the rationale, aims, and context of a study. Questions 4-8 relate to the rigor of the research methods that were used to establish the trustworthiness of the findings. Questions 9 and 10 assess the credibility of study methodologies and help ensure that that the findings of different studies are valid and meaningful. Finally, question 11 assesses the relevance of a study to the software engineering research community and industry.

Evaluating each potential primary study based on these 11 criteria provides a measure of the extent to which the findings of a particular study can make a valuable contribution to this SLR. Each quality question is either answered as yes (1 point) or no (0 points), so the quality score of a study ranges between 0 (Very poor) to 11 (very good).

5.5 Data Extraction

A predefined extraction form was used to obtain relevant data from each of the 12 primary studies included in the SLR. Relevant data includes what FMs or tools are used in a study, what domain the study is conducted in, and how useful the approaches are in practice. The data extraction form is an important tool that enables the recording of the full details of the studies under review and the details on how each study addresses the research questions of this SLR. In addition, the data extraction form is a useful tool for organizing information from a large number of studies.

Table 1. Quality criteria for selection of primary studies (adapted from [18])

Questions
1. Is the paper based on empirical research, rather than a “lessons learned" report based on expert opinion?
2. Are the aims of the research clearly stated?
3. Was the research environment and context of the research adequately described?
4. Was the research design appropriate to address the aims of the research?
5. Was the recruitment strategy appropriate to the aims of the research?
6. Was there a control group with which to compare treatments?
7. Was the data collected in a way that addressed the research issue?
8. Was the data analysis sufficiently rigorous?
9. Has the relationship between researcher and participants been adequately considered?
10. Is there a clear statement of findings?
11. Is the study of value for research or practice?

6 Results

This section reports the results of the SLR based on the research questions defined in Section 5.1.

6.1 Research Question 1

“What types of FMs and tools are used for specifying and verifying industrial software systems?"

Two studies report report on the use the Analytical Software Design (ASD) method at Philips Healthcare [21, 45]. ASD is a component-based, model-driven technology that combines the application of FMs (e.g., Sequence-Based Specification (SBS) [14], Communicating Sequential Processes (CSP) [25], and the model checker Failure Divergence Refinement (FDR2) [1]) with various software development methods, including the Box Structure Development Method (BSDM), Stepwise Refinement, and Component-Based Software Development [44]. The ASD approach is particularly useful for developing complex control software and event-driven systems where concurrency plays a crucial role. An ASD specification is written with SBS and is used to build models of the functional behavior of ASD components. Using ASD models, formal models like CSP and source code

implementations (written in C++ or C#) can be automatically generated. Via this mechanized transformation, ASD guarantees consistency between specification, verified models, and implementation code of ASD components [44].

Newcombe et. al [42] delineate that engineers at Amazon Web Services use TLA+ for writing formal specifications. TLA+ is a formal specification language based on basic set theory and predicates. The TLA+ specification language either uses conventional mathematical reasoning or the TLC model checker to show that a system design correctly implements the desired correctness properties. One of the main benefits of using TLA+ is that it can be used to write precise, formal descriptions of almost any kind of discrete system [31]. Engineers at Amazon Web Services also use PlusCal, a formal specification language that can be automatically translated to TLA+ with the press of a single key. PlusCal is intended to be a direct replacement for pseudo-code and resembles an imperative programming language (similar to C-style programming languages) that uses TLA+ for expressions and values. PlusCal is a more user-friendly language that requires no prior familiarity with TLA+. However, TLA+ provides greater flexibility over PlusCal, where developers have the freedom to choose and adjust different levels of abstraction.

Rodrigues et. al [50] focus on the use of Z notation for formal specification in Agile projects. The Z notation is based upon set theory (i.e., standard set operators, set comprehensions, Cartesian products, and power sets) and mathematical logic (i.e., first-order predicate calculus) [64]. Z uses *schemas*, or patterns of declaration and constraint, to structure mathematical objects and their properties. Every object in the Z notation has a unique *type* that is represented as a maximal set in the current specification. In combination with natural language, Z can be used to write formal specifications. Z is not intended for the description of non-functional properties (i.e., usability, performance, size, and reliability), nor is it intended for the description of timed or concurrent behavior [64].

Bennion and Habli [7] consider the use of Simulink Design Verifier (SDV), a model checker that forms part of a modelling system already widely used in the safety-critical industry. They investigate if SDV can provide a practical route to effective formal verification. The study also considers the extent to which model checking can satisfy the requirements of the DO-178C guidance on formal methods.

Post et. al [48] investigate if real-time specification patterns (RTSP) (proposed by Konrad and Cheng [29]) suffices to formally express behavioral requirements taken from automotive projects at BOSCH. RTSP is represented in a restricted English grammar that can be automatically translated to logics, but looks like natural language. Filipovikj et. al [20] also focus on the use of the RTSP in the automotive domain. This set of patterns was chosen because it provides a quantitative notion of time, which is needed for the formalization of real-time requirements in the automotive domain.

Arun et. al [2] propose using the Python programming language as a formal specification tool. The authors note that the number of successes of using FMs in industry remains very low because it is difficult to learn new and complex mathematical notations, there is a lack of experts on FMs, and it is difficult to review large specifications that are written in mathematical notations. One of the benefits of Python is that it is an interpreted programming language with expressive syntax that some have compared to executable pseudocode [43]. Thus, the authors investigate if using Python as a formal specification language can be effective and solve the problems that are associated with traditional FMs.

Moy et. al [38] present a multi-case study of two companies in the avionics domain. One case focuses on the use of unit proof for verifying functional properties. This is a program proof method based on CAVEAT, a C program prover developed by the commissariat à l'énergie atomique, and is applied in the verification process of a safety-critical avionics program to achieve DO-178 objectives [54]. These objectives relate to verifying that the executable code meets the functional LLRs.

Ferrari et. al [19] report on the use of a formal development approach that integrates SysML and Simulink/Stateflow. SysML (Systems Modeling Language) is a general-purpose architecture modeling language for system development [56]. Stateflow provides users with a graphical language that includes state transition diagrams, flow charts, state transition tables, and truth tables. Stateflow can be used to describe how MATLAB algorithms and Simulink models react to input signals, events, and time-based conditions [35].

Sune Wolff [62] focuses on the use of Vienna Development Method (VDM) for specifying and developing software in the avionics domain. VDM consists of a group of mathematically well-founded languages and tools for expressing and analyzing system models during early design stages, before expensive implementation commitments are made. The construction and analysis of the model help to identify areas of incompleteness or ambiguity in informal system specifications, and provide some level of confidence that a valid implementation will have key properties, especially those of safety or security [40].

Bozzano et. al [11] use AADL models and the COMPASS toolset for validating spacecraft designs. The COMPASS toolset uses a coherent set of specification and analysis techniques to evaluate the system-level correctness, safety, dependability and performability of on-board computer-based aerospace systems [16]. COMPASS uses the SLIM language, a dialect based on AADL, to generate its input models. This makes it possible to describe both the hardware and software components of the system, and their connections. Separate

error models can be defined to describe faults, which can automatically be injected into the model [16].

6.2 Research Question 2

"How useful are FMs and tools when applied to industrial software systems?"

Groote et. al [21] report on their experience of applying the ASD method to the development of the Front-end client (FEClient) software unit of highly sophisticated X-ray machines at Philips Healthcare. The main responsibility of the FEClient is to guard the flow of information between two concurrent subsystemes of the X-ray machines. The control part of the FEClient was developed in a pipeline of consecutive increments, where each increment was described using an ASD specification and mathematically verified via model checking. Throughout this process, only eleven coding errors were detected during the construction of 28 thousand lines of code. Team members credit the ultimate quality of the FEClient software to the rigor and formal technologies supplied by the ASD method, including specification reviews, formal behavioral verification, and model checking.

Osaiweran et. al [45] also use the ASD approach at Philips Healthcare. In this case, the authors combined the use of the commercial tool ASD:Suite and test-driven development to develop the power control service of an interventional X-ray system. The end quality of the power control service was very high, exhibiting only 0.17 defects per KLOC. This level of quality compares favorably with the industry standard defect rate of 1-25 defects per KLOC [36]. Further testing was conducted by independent testing teams and no errors were found.

Amazon engineers used TLA+ during the development of 10 large complex real-world systems and report that TLA+ added significant value in all cases [42]. Engineers credit the use of TLA+ to finding subtle bugs that would not have been found by other means. Moreover, TLA+ provided engineers with enough understanding and confidence to make aggressive performance optimizations without sacrificing program correctness. Compared to traditional proof writing, writing formal TLA+ specification was more reliable and less time consuming. This suggests an improvement in time to market and a good return on investment.

Rodrigues et. al [50] conducted a survey to collect the opinions of practitioners on using FMs for requirements specification in Agile projects. The most cited advantages of using formal specification are standardization of requirements writing, eliminating ambiguity, and producing consistent and precise specifications. Survey results also indicate that the main challenges in applying FMs to requirements specification are related to the complexity of understanding formal specification (e.g., formal specification demands more time, is more complex than informal specification, and requires training). The authors also conducted a multi-case study to determine how formal specification (particularly the Z notation) can contribute to Agile projects in the business domain. They report that formal specification helps remove specification errors and works better for complex problems. However, the main limitations are lack of knowledge about formal specification and the amount of time needed for formal specification.

Bennion and Habli [7] report that model checking with SDV can find errors earlier in the development cycle than traditional verification. This has the potential to save time and money due to reduced scrap and rework. However, no evidence was found that model checking finds more errors than traditional verification. The authors also find that the benefits of SDV can be realized in an industrial setting without specialist skills.

Post et. al [48] conclude that SPS suffices to express automotive behavioral requirements at BOSCH. They found that a majority of traditional requirements could be reformulated in the SPS. Only a few patterns were needed to express most automotive behavioral requirements at BOSCH, but more studies with requirements of automotive suppliers are needed to refute or strengthen this belief for the entire automotive domain.

Filipovikj et. al [20] investigate if SPS can be used to transform system requirements for Scania Electrical and Electronic (E/E) systems inside heavy road vehicles from their traditional written form into semi-formal notation. One of the reported benefits of using SPS is an apparent reduction of ambiguity in the system requirements. Scania engineers also report that using SPS provides a useful support structure for discussing the meaning of certain requirements, and this is expected to have a positive impact on communication between stakeholders. Moreover, SPS is applicable to industrial settings because it improves the testability of the requirements and the patterning process was conducted in a reasonable time frame. The authors conclude that the concept of patterns is likely to be generally applicable to the automotive domain.

Arun et. al [2] use Python for writing formal specifications of software in nuclear industry. They identify that using Python increases usability, reviewability, and scalability of formal specifications. High level languages like Python do not require special tools or a strong mathematical background to write specifications. The simple syntax of Python can also significantly reduce the effort needed to maintain the specifications. Moreover, the syntax of Python resembles natural language, which simplifies the review process for interested stakeholders (i.e., managers, customers, and certifiers).

Moy et. al [38] focus on software systems for commercial aircrafts at two companies: Dassault-Aviation and Airbus. Both companies successfully applied formal verification early on in the software development process. The evidence suggests that formal verification is practical in a DO-178 context and can be a cost-effective alternative to testing. For example,

the ambiguities of natural language can be avoided when formulating requirements with a formal notation. Formal analysis techniques can then be used to check for consistency, and this is especially useful because in practice, most errors stem from the requirements instead of the code.

Ferrari et. al [19] reports a 10 year experience of how General Electric Transportation Systems (GETS), a medium-sized branch of a global railway signaling manufacturer, adopted general purpose, model-based tools aided by formal methods to develop its products. Initially, developers at GETS used the models designed through Simulink/Stateflow solely for requirements elicitation. However, the company also wanted to explore the use of these models for code generation. Thus, GETS adopted model-based testing and abstract interpretation, as well as language restrictions to reduce the tool suite's semi-formal semantics to a formal semantics. This model-based approach sped up development and allowed the company to handle more complex systems.

One study [62] developed an executable model of a self-defense system for a fighter aircraft using VDM. A large test suite of scenario-based tests was used to exercise the model. An analysis of the resulting output lead to a lot of insight being gained regarding the general functionality of the system, along with the new interpretation of the messages passed between ECAP and the AS subsystem. This proved to be a valuable asset in reaching an agreement with the customer, where the customer was particularly impressed by the extensive tests which had been carried out on the model. The log files generated by running tests served as an important tool for facilitating effective communication between the customer and the systems engineers in charge of the project.

Bozzano et. al [11] use the COMPASS toolset for validating spacecraft designs. The authors conducted three pilot projects over a span of five years that focused on system-level and subsystem-level designs of past and ongoing space missions. These projects focused on the definition and analysis of extremely large spacecraft systems and resulted in an advancement of validating spacecraft designs for correctness, safety, dependability, and performance. Moreover, the software readiness level increased from level 1 to early level 4.

7 Discussion

The results of this SLR suggest that a wide range of FMs can be useful for developing industrial software systems. FMs appear to be particularly useful for developing safety-critical systems, where defects in the software can potentially result in the loss of human lives. Every primary study focused on the application of FMs and tools in safety-critical domains, such as aviation (e.g., military jets and spacecrafts), automotive, railway, healthcare, and nuclear. Moreover, many studies report that the application of FMs had a positive effect on the end-product, such as increased software quality. Other positive effects include an increase in productivity during the software development process, reductions in ambiguity, and an increase in knowledge transfer.

Moreover, a number of the studies from the SLR credit the successful application of FMs to the technological advancements in FM tools. These tools reduce the time needed to learn and understand the complex mathematics behind FMs. Early evidence suggests that even developers with no prior FMs experience can successfully apply FM tools to real-world projects. As tools become more user friendly, a rise in the adoption of FMs in industry is expected. A 2020 report by Margaria and Kinry identifies that the current usage of FMs in industry is rising due to technological advancements in FM tools [34]. This has driven industry giants, such as Apple, Amazon, Google, and NASA, to adopt FMs in daily business practices.

Although the results of this SLR suggest that FMs can be useful in industry, one of the reported drawbacks of FMs stems from the learning time that is required to become comfortable with these methods and the tools that support them. Many studies report that engineers require some time to get acclimated to the FMs and tools that are deployed at the beginning of the project cycle. However, as the project progresses, the engineers became more comfortable and productive. This is a common occurrence whenever a new method or technology is adopted at a company. For example, one of the drawbacks of using pair programming in industry is that it takes time, usually around a week, before a pair becomes productive (also known as *pair jelling*) [60]. The results of this SLR suggest that a similar phenomenon applies to FMs, where productivity gradually increases as a person learns how to apply a particular method or tool to his or her respective work domain.

One area of concern is that no studies report negative experiences when applying FMs to the development process of industrial software systems. There are at least two possible explanations for this: either FMs truly are a superior technique for producing quality software with minimal defects, or researchers do not want to publish negative experiences when using FMs. If FMs are a truly superior technique, then the adoption of these techniques should be a common practice in industry. Some primary studies do mention that FMs and tools are being rapidly adopted as more user-friendly tools are being invented and deployed in industry. This suggests a recent rise in the industrial use of FMs, but finding published studies written within the 2010-20 timeframe was a difficult task. An overwhelming majority of the studies from this timeframe either propose new FMs or apply FMs to small-scale software systems in experimental settings. This indicates a need for more research that investigates the usefulness of FMs when applied in industry.

8 Limitations

Several factors need to be considered when generalizing the results of this SLR. First, only one person (the author) was responsible for carrying out the entire review process. The author defined the review protocol and carried out the major tasks involved in each phase of the SLR, which may have unwittingly biased the results. For example, having only one person to select primary studies based on a set of inclusion and exclusion criteria could have resulted in the inclusion of studies that should have been omitted (or vice versa). However, an effort was made to avoid bias by closely following the recommendations suggested in the SLR guidelines [3, 28].

Another limiting factor stems from the process of identifying relevant literature, where primary studies were only obtained from electronic sources. Without searching non-electronic conference proceedings and journals, there is potential that relevant studies were omitted from this SLR. Moreover, only papers that are written in the languages of English or Polish were considered during the primary study selection process. This is due to the fact that the author can only read in these two languages. However, papers written in other languages could have been translated into English using a tool like Google Translate, and thus resulted in the discovery of relevant studies and key insights about FMs.

Another exclusion criteria poses a limitation because it excludes studies that use FMs in experimental settings. Experimental studies are important because they are replicable and provide researchers with an opportunity to gain quantitative results that are more statistically significant than industrial studies, which tend to only be one data point. Experiments also enable the use of a control group, whereas asking two teams to perform identical tasks in an industrial setting is impractical and expensive. Despite their valuable aspects, experimental studies were excluded because this SLR focuses on the use of FMs for the development of industrial software systems. Compared to an experiment where subjects are asked to develop small-scale systems in a short period of time, an industrial domain features larger and more complex software systems that can take months, or even years, to develop. Since it is impractical, if not impossible, to develop an application of industrial size and complexity in an experiment, it was necessary to exclude these studies from the SLR.

9 Future Work

One of the benefits of FMs is that they help produce high quality software with minimal defects. A similar benefit is reported for the use of pair programming (PP) when developing commercial software applications [6, 17, 23, 55, 57]. This poses an opportunity to investigate if combining the use of PP and FMs can improve software quality more than using FMs alone. In addition, both FMs and PP require some acclimation time before becoming effective. Perhaps the addition of an extra developer that comes from PP may result in a faster acclimation time to FMs than a single person who is using FMs. Combining PP and FMs may also help facilitate knowledge transfer throughout a team. However, PP carries the inherent burden of an increase in cost with the addition of an extra developer. Thus, it is important to conduct research that investigates the costs and benefits (both economic and social) of combining FMs and PP in industrial domains. This research will result in credible evidence that companies can use when thinking about adopting the novel technique.

Another avenue for future work will focus on taking the results of this SLR and comparing them to a wide range of studies that use the same FMs and tools in experimental settings. It will be interesting to see if similar benefits and drawbacks are observed in experimental environments. If there are differences, it will be worthwhile to investigate how two or more studies that use the same method or tool come up with different results. Perhaps certain factors, such as the complexity of a given programming task, may influence the success or failure of applying a particular method or tool. Identifying these factors is an important area of research because knowing them in advance will lead to the more effective deployment of FMs and tools.

10 Conclusion

The results of this SLR suggest that a wide variety of FMs and tools are being successfully applied in industry. As commercial pressure to produce higher quality software continues to increase, FMs have an opportunity to play a crucial role in the development of safety-critical software systems. Tools that support FMs are becoming easier to use, while yielding the same or better quality benefits compared to applying traditional FMs. In a world where market pressure also plays a major role in getting a system developed on time, it is important that practitioners are sufficiently trained in the FMs or tools that they are applying. Moreover, the FMs research community needs to continue to invest effort into investigating the usefulness of FMs in industry. The more evidence that exists about the successful application of FMs and tools, the more likely that practitioners in industry will adopt them.

Acknowledgments

I would like to express my very great appreciation to Dr. Mei Nagappan for his valuable and constructive suggestions during the planning and development of this research work. I would also like to thank David Radke (PhD candidate, University of Waterloo) for his help during the revision process of this paper.

References

[1] 2012. Failures-divergence Refinement Fdr2 User Manual.

[2] B. P. Arun, N. Murali, P. Swaminathan, and K. C. Senthil. 2010. Making formal software specification easy. In *2010 2nd International Conference on Reliability, Safety and Hazard - Risk-Based Technologies and Physics-of-Failure Methods (ICRESH)*. 511–516.

[3] Kitchenham BA and Stuart Charters. 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. 2 (01 2007).

[4] J. Backes, P. Bolignano, B. Cook, A. Gacek, K. S. Luckow, N. Rungta, M. Schaef, C. Schlesinger, R. Tanash, C. Varming, and M. Whalen. 2019. One-Click Formal Methods. *IEEE Software* 36, 6 (2019), 61–65.

[5] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 1994. The Goal Question Metric Approach.

[6] Andrew Begel and Nachiappan Nagappan. 2008. Pair Programming: What's in it for Me? 120–128. https://doi.org/10.1145/1414004.1414026

[7] Matthew Bennion and Ibrahim Habli. 2014. A Candid Industrial Evaluation of Formal Software Verification Using Model Checking. In *Companion Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 175–184. https://doi.org/10.1145/2591062.2591184

[8] Per Bjesse. 2005. What is Formal Verification? *SIGDA Newsl.* 35, 24 (Dec. 2005), 1–es. https://doi.org/10.1145/1113792.1113794

[9] J. P. Bowen and M. G. Hinchey. 1995. Seven more myths of formal methods. *IEEE Software* 12, 4 (1995), 34–41.

[10] Jonathan P. Bowen and Michael G. Hinchey. 1995. Ten Commandments of Formal Methods. *Computer* 28, 4 (April 1995), 56–63. https://doi.org/10.1109/2.375178

[11] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Panagiotis Katsaros, Konstantinos Mokos, Viet Yen Nguyen, Thomas Noll, Bart Postma, and Marco Roveri. 2014. Spacecraft early design validation using formal methods. *Reliability Engineering System Safety* 132 (2014), 20 – 35. https://doi.org/10.1016/j.ress.2014.07.003

[12] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. 2007. Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain. *J. Syst. Softw.* 80, 4 (April 2007), 571–583. https://doi.org/10.1016/j.jss.2006.07.009

[13] Ricky Butler. 2016. *What is Formal Methods?* NASA. https://shemesh.larc.nasa.gov/fm/fm-what.html.

[14] Jason M. Carter and Jesse H. Poore. 2007. Sequence-Based Specification of Feedback Control Systems in Simulink®. In *Proceedings of the 2007 Conference of the Center for Advanced Studies on Collaborative Research* (Richmond Hill, Ontario, Canada) *(CASCON '07)*. IBM Corp., USA, 332–345. https://doi.org/10.1145/1321211.1321257

[15] Edmund M. Clarke and Jeannette M. Wing. 1996. Formal Methods: State of the Art and Future Directions. *ACM Comput. Surv.* 28, 4 (Dec. 1996), 626–643. https://doi.org/10.1145/242223.242257

[16] COMPASS 2020. *COMPASS: Correctness, Modeling and Performance of Aerospace Systems.* COMPASS. http://www.compass-toolset.org/.

[17] E. di Bella, I. Fronza, N. Phaphoom, A. Sillitti, G. Succi, and J. Vlasenko. 2013. Pair Programming and Software Defects–A Large, Industrial Case Study. *IEEE Transactions on Software Engineering* 39, 7 (2013), 930–953.

[18] Tore Dybå and Torgeir Dingsøyr. 2008. Empirical Studies of Agile Software Development: A Systematic Review. *Inf. Softw. Technol.* 50, 9–10 (Aug. 2008), 833–859. https://doi.org/10.1016/j.infsof.2008.01.006

[19] A. Ferrari, A. Fantechi, S. Gnesi, and G. Magnani. 2013. Model-Based Development and Formal Methods in the Railway Industry. *IEEE Software* 30, 3 (2013), 28–34.

[20] P. Filipovikj, M. Nyberg, and G. Rodriguez-Navas. 2014. Reassessing the pattern-based approach for formalizing requirements in the automotive domain. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 444–450.

[21] Jan Friso Groote, Ammar Osaiweran, and Jacco Wesselius. 2012. Experience Report on Developing the Front-End Client Unit under the Control of Formal Methods. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (Trento, Italy) *(SAC '12)*. Association for Computing Machinery, New York, NY, USA, 1183–1190. https://doi.org/10.1145/2245276.2231962

[22] A. Hall. 1990. Seven myths of formal methods. *IEEE Software* 7, 5 (1990), 11–19.

[23] Gerard Hartnett and Brian Fitzgerald. 2005. A Study of the Use of Agile Methods within Intel. *International Federation for Information Processing Digital Library; Business Agility and Information Technology Diffusion;* 180. https://doi.org/10.1007/0-387-25590-7_12

[24] Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy Vilkomir, Martin R. Woodward, and Hussein Zedan. 2009. Using Formal Specifications to Support Testing. *ACM Comput. Surv.* 41, 2, Article 9 (Feb. 2009), 76 pages. https://doi.org/10.1145/1459352.1459354

[25] C. A. R. Hoare. 1978. Communicating Sequential Processes. *Commun. ACM* 21, 8 (Aug. 1978), 666–677. https://doi.org/10.1145/359576.359585

[26] Christoph Kern and Mark R. Greenstreet. 1999. Formal Verification in Hardware Design: A Survey. *ACM Trans. Des. Autom. Electron. Syst.* 4, 2 (April 1999), 123–193. https://doi.org/10.1145/307988.307989

[27] Khalid S. Khan, Regina Kunz, Jos Kleijnen, and Gerd Antes. 2011. Systematic reviews to support evidence-based medicine.

[28] Barbara Kitchenham. 2004. Procedures for Performing Systematic Reviews. *Keele, UK, Keele Univ.* 33 (08 2004).

[29] Sascha Konrad and Betty H. C. Cheng. 2005. Real-Time Specification Patterns. In *Proceedings of the 27th International Conference on Software Engineering* (St. Louis, MO, USA) *(ICSE '05)*. Association for Computing Machinery, New York, NY, USA, 372–381. https://doi.org/10.1145/1062455.1062526

[30] Subodh Kumar. 2013. Formal methods for software specification and analysis:An Overview.

[31] Leslie Lamport. 2002. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley Longman Publishing Co., Inc., USA.

[32] Axel van Lamsweerde. 2000. Formal Specification: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (Limerick, Ireland) *(ICSE '00)*. Association for Computing Machinery, New York, NY, USA, 147–159. https://doi.org/10.1145/336512.336546

[33] D. Mandrioli. 2015. On the Heroism of Really Pursuing Formal Methods. In *2015 IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering*. 1–5.

[34] T. Margaria and J. Kiniry. 2020. Welcome to Formal Methods in Industry. *IT Professional* 22, 1 (2020), 9–12.

[35] MATLAB & Simulink 2020. *Stateflow: Model and simulate decision logic using state machines and flow charts.* MATLAB & Simulink. https://www.mathworks.com/products/stateflow.html.

[36] Steve McConnell. 2004. *Code complete, second edition.* Microsoft Press.

[37] J. B. Michael, G. W. Dinolt, and D. Drusinsky. 2020. Open Questions in Formal Methods. *Computer* 53, 5 (2020), 81–84.

[38] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate. 2013. Testing or Formal Verification: DO-178C Alternatives and Industrial Experience. *IEEE Software* 30, 3 (2013), 50–57.

[39] Y. Murray and D. A. Anisi. 2019. Survey of Formal Verification Methods for Smart Contracts on Blockchain. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 1–6.

[40] Andreas Müller. 2009. VDM — The Vienna Development Method. (04 2009).

[41] S. P. Nanda and E. S. Grant. 2019. A Survey of Formal Specification Application to Safety Critical Systems. In *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*. 296–302.

[42] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon Web Services Uses Formal Methods. *Commun. ACM* 58, 4 (March 2015), 66–73. https://doi.org/10.1145/2699417

[43] T. E. Oliphant. 2007. Python for Scientific Computing. *Computing in Science Engineering* 9, 3 (2007), 10–20.

[44] A.A.H. Osaiweran, M. Boosten, and M.R. Mousavi. 2010. *Analytical software design : introduction and industrial experience report.* Technische Universiteit Eindhoven.

[45] Ammar Osaiweran, Mathijs Schuts, Jozef Hooman, and Jacco Wesselius. 2013. Incorporating Formal Techniques into Industrial Practice: an Experience Report. *Electronic Notes in Theoretical Computer Science* 295 (2013), 49 – 63. https://doi.org/10.1016/j.entcs.2013.04.005 Proceedings the 9th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA).

[46] D. L. Parnas. 2010. Really Rethinking 'Formal Methods'. *Computer* 43, 1 (2010), 28–34.

[47] S. L. Pfleeger and L. Hatton. 1997. Investigating the influence of formal methods. *Computer* 30, 2 (1997), 33–43.

[48] Amalinda Post, Igor Menzel, Jochen Hoenicke, and Andreas Podelski. 2012. Automotive Behavioral Requirements Expressed in a Specification Pattern System: A Case Study at BOSCH. *Requir. Eng.* 17, 1 (March 2012), 19–33. https://doi.org/10.1007/s00766-011-0145-9

[49] C. Rodrigues. 2009. A case study for Formal Verification of a timing co-processor. In *2009 10th Latin American Test Workshop*. 1–6.

[50] Peterson Rodrigues, Miguel Ecar, Stefane V. Menezes, João Pablo S. da Silva, Gilleanes T. A. Guedes, and Elder M. Rodrigues. 2018. Empirical Evaluation of Formal Method for Requirements Specification in Agile Approaches. In *Proceedings of the XIV Brazilian Symposium on Information Systems* (Caxias do Sul, Brazil) *(SBSI'18)*. Association for Computing Machinery, New York, NY, USA, Article 53, 8 pages. https://doi.org/10.1145/3229345.3229401

[51] K. Schaffer and J. Voas. 2016. What Happened to Formal Methods for Security? *Computer* 49, 8 (2016), 70–79.

[52] R. Sinha, S. Patil, L. Gomes, and V. Vyatkin. 2019. A Survey of Static Formal Methods for Building Dependable Industrial Automation Systems. *IEEE Transactions on Industrial Informatics* 15, 7 (2019), 3772–3783.

[53] A. E. K. Sobel and M. R. Clarkson. 2002. Formal methods application: an empirical tale of software development. *IEEE Transactions on Software Engineering* 28, 3 (2002), 308–320.

[54] Jean Souyris and Denis Favre-Félix. 2004. Proof of Properties in Avionics. In *Building the Information Society*, Renè Jacquart (Ed.). Springer US, Boston, MA, 527–535.

[55] Wenying Sun, George Marakas, and Miguel Aguirre-Urreta. 2015. Effectiveness Of Pair Programming: Perceptions Of Software Professionals. *IEEE Software* 33 (01 2015), 1–1. https://doi.org/10.1109/MS.2015.106

[56] SysML.org 2020. *SysML Open Source Project - What is SysML? Who created it?* SysML.org. https://sysml.org/.

[57] Jari Vanhanen and Casper Lassenius. 2007. Perceived Effects of Pair Programming in an Industrial Context. *Conference Proceedings of the EUROMICRO*, 211 – 218. https://doi.org/10.1109/EUROMICRO.2007.47

[58] A. Wassyng. 2013. Though this be madness, yet there is method in it? (Keynote). In *2013 1st FME Workshop on Formal Methods in Software Engineering (FormaliSE)*. 1–7.

[59] Danny Weyns, M. Usman Iftikhar, Didac Gil de la Iglesia, and Tanvir Ahmad. 2012. A Survey of Formal Methods in Self-Adaptive Systems. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering* (Montreal, Quebec, Canada) *(C3S2E '12)*. Association for Computing Machinery, New York, NY, USA, 67–79. https://doi.org/10.1145/2347583.2347592

[60] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries. 2000. Strengthening the case for pair programming. *IEEE Software* 17, 4 (2000), 19–25.

[61] J. M. Wing. 1990. A specifier's introduction to formal methods. *Computer* 23, 9 (1990), 8–22.

[62] Sune Wolff. 2015. Using Executable VDM++ Models in an Industrial Application - Self-defense System for Fighter Aircraft.

[63] Yaron Wolfsthal and Rebecca M. Gott. 2005. Formal Verification: Is It Real Enough?. In *Proceedings of the 42nd Annual Design Automation Conference* (Anaheim, California, USA) *(DAC '05)*. Association for Computing Machinery, New York, NY, USA, 670–671. https://doi.org/10.1145/1065579.1065755

[64] Jim Woodcock and Jim Davies. 1996. *Using Z: Specification, Refinement, and Proof.* Prentice-Hall, Inc., USA.

[65] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal Methods: Practice and Experience. *ACM Comput. Surv.* 41, 4, Article 19 (Oct. 2009), 36 pages. https://doi.org/10.1145/1592434.1592436