New Opportunities for Integrated Formal Methods[1]

A Preprint

Mario Gleirscher, Simon Foster, and Jim Woodcock

Department of Computer Science, University of York, United Kingdom
Deramore Lane, Heslington, York YO10 5GH

November 6, 2019

Abstract

Formal methods have provided approaches for investigating software engineering fundamentals and also have high potential to improve current practices in dependability assurance. In this article, we summarise known strengths and weaknesses of formal methods. From the perspective of the assurance of *robots and autonomous systems* (RAS), we highlight new opportunities for integrated formal methods and identify threats to the adoption of such methods. Based on these opportunities and threats, we develop an agenda for fundamental and empirical research on integrated formal methods and for successful transfer of validated research to RAS assurance. Furthermore, we outline our expectations on useful outcomes of such an agenda.

***K*eywords** Formal methods · strengths · weaknesses · opportunities · threats · SWOT · challenges · integration · unification · research agenda · robots and autonomous systems

Acronyms

AI	artificial intelligence
ASIL	automotive safety integrity level
DA	dependability assurance
DAL	design assurance level
DSE	dependable systems engineering
DSL	domain-specific language
FDA	Food and Drug Administration
FI	formal inspection
FM	formal method
HCI	human-computer interaction
iFM	integrated formal method
IT	information technology
MBD	model-based development
MDE	model-driven engineering
ML	machine learning
RAS	robots and autonomous systems
RCA	root cause analysis
RE	requirements engineering
SACM	Structured Assurance Case Meta-model
SC	systematic capability
SIL	safety integrity level
SMT	Satisfiability Modulo Theory
SWOT	strengths, weaknesses, opportunities, and threats
SysML	Systems Modelling Language
UML	Unified Modelling Language
UTP	Unifying Theories of Programming

1 Introduction

A plethora of difficulties in software practice and momentous software faults have been continuously delivering reasons to believe that a significantly more rigorous discipline of software engineering is needed [101]. Researchers such as Neumann [141] have collected plenty of anecdotal evidence on software-related risks substantiating this belief.

In *dependable systems engineering*, researchers have turned this belief into one of their working hypotheses and contributed formalisms, techniques, and tools to increase the rigour in engineering workflows [101]. Examples of activities where formalisms have been brought to bear include requirements engineering [e.g. 71], architecture design and verification, test-driven development, program synthesis, and testing, to name a few. *Formal methods (FMs)* have

[1]Partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the Grant no. 381212925. © 2019. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in ACM Computing Surveys, https://doi.org/10.1145/3357231.

*correspondence: mario.gleirscher@york.ac.uk

been an active research area for decades, serving as powerful tools for *theoretical researchers*. As a paradigm to be adopted by *practitioners* they have also been investigated by *applied researchers*. Applications of FMs have shown their strengths but also their weaknesses [101, 125].

Based on observations contradicting the mentioned difficulties, Hoare [88] came to ask "How did software get so reliable without proof?" He, and later MacKenzie [125], suggested that continuously improved design of programming languages and compilers, defensive programming, inspection, and testing must have effectively prevented many dangerous practices. MacKenzie coined the "Hoare paradox" stating that although proof was seldom used, software had shown to be surprisingly fit for purpose. However, faced with software of increasing complexity (e.g. RAS), MacKenzie pondered how long Hoare's question will remain valid.

Indeed, recently we can observe a plethora of difficulties with robots and autonomous systems [112, 141]. Such systems are set to be more broadly deployed in society, thereby increasing their level of safety criticality [70] and requiring a stringent regulatory regime. A successful method for regulatory acceptance is provided by structured assurance cases, which provide comprehensible and indefeasible safety arguments supported by evidence [72, 77, 104]. However, such assurance cases—whether or not compliant with standards like IEC 61508[1] and DO-178C[2]—can be laborious to create, complicated to maintain and evolve, and must be rigorously checked by the evaluation process to ensure that all obligations are met and confidence in the arguments is achieved [68, 160]. Nevertheless, these are problems that FMs are designed to overcome.

In spite of the weaknesses of current FMs, and encouraged by their strengths, we believe that their coordinated use within established processes can reduce critical deficits observable in dependable systems engineering. Farrell et al. [45] state that "there is currently no general framework integrating formal methods for robotic systems". The authors highlight the use of what are called *integrated formal methods (iFMs)*[3] in the construction of assurance cases and the production of evidence as a key opportunity to meet current RAS challenges. Particularly, computer-assisted assurance techniques [172], supported by evidence provided by iFMs, can greatly increase confidence in the sufficiency of assurance cases, and also aid in their maintenance and evolution through automation. Moreover, the use of modern FM-based tools to support holistic simulation, prototyping, and verification activities, at each stage of system, hardware, and software development, can lead to systems that are demonstrably safe, secure, and trustworthy.

1.1 Contribution

We investigate the *potentials for the wider adoption of integrated formal methods in dependable systems engineering*, taking robots and autonomous systems as a recent opportunity to foster research in such methods and to support their successful transfer and application in practice.

We analyse the *strengths, weaknesses, opportunities, and threats* of using formal methods in the assurance of dependable systems such as RAS. For this analysis, we summarise experience reports on formal method transfer and application.

Assuming that (a) integration enhances formal methods and (b) the assurance of robots and autonomous systems is a branch of dependable systems engineering, Figure 1 shows how we derive an agenda for fundamental and applied iFM research.

From the *strengths*, we see in recent research, and from the *opportunities* in current RAS assurance, we argue why this domain is a key opportunity for iFMs. Particularly, we indicate how such methods can meet typical assurance challenges that RASs are increasingly facing.

From the *weaknesses*, we observe in recent research, and from the *threats* formal method research transfer is exposed to, we derive directions of foundational and empirical research to be taken to transfer iFMs into the assurance of robots, autonomous systems, and other applications, and to use these methods to their maximum benefit.

Our analysis 1. elaborates on the analysis and conclusions of Hoare et al. [90], 2. extends their suggestions with regard to formal method experimentation and empirical evidence of effectiveness focusing on collaboration between formal method researchers and practitioners, and 3. develops a research and research transfer road map, placing emphasis on RASs.

[1]For: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems.
[2]For: Software Considerations in Airborne Systems and Equipment Certification.
[3]We reuse the term from the homonymous conference series [4].

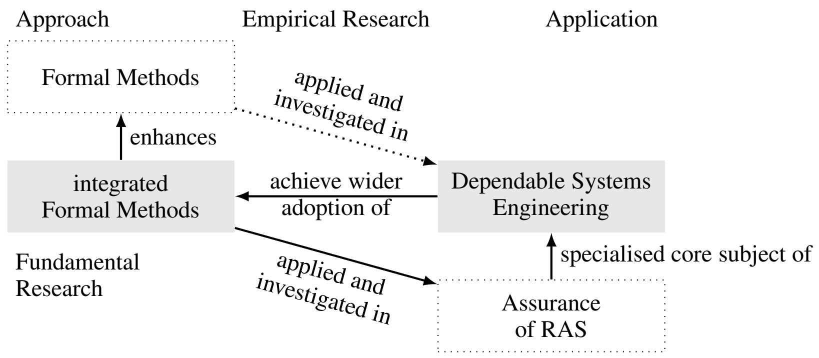
Approach
Empirical Research
Application
Formal Methods
applied and investigated in
enhances
integrated Formal Methods
achieve wider adoption of
Dependable Systems Engineering
Fundamental Research
applied and investigated in
specialised core subject of
Assurance of RAS

Figure 1: RAS assurance as an opportunity for the wider adoption of iFMs in dependable systems practice

1.2 Overview

We provide some background including terminology (Section 2.1) and related work (Section 2.2) in the following. Then, we carry through an analysis of strengths, weaknesses (Section 3), opportunities (Section 4), and threats (Section 5) of iFMs for RAS assurance. Based on this analysis, we formulate our hypotheses (Section 6), pose research questions based on these hypotheses, derive a research agenda, and specify some outcomes we expect from this agenda (Section 7).

2 Background

This section introduces the core terminology used throughout this article, and provides a discussion of other surveys of the FM state of the art and a summary of similar positions and agendas.

2.1 Terminology

For the sake of clarity among readers of different provenance, we restrict the meaning of some terms we use in the following and introduce convenient abbreviations.

We view *robots and autonomous systems* as both dependable systems and highly automated machines capable of achieving a variety of complex tasks in support of humans. We can consider such systems by looking at four layers: the plant or process composed of the operational environment and the machine; the machine itself; the machine's controller, and the software embedded into this controller. Based on these layers, we treat "embedded system" and "embedded software" as synonyms. Machine, controller, and software can all be distributed.

By *dependable systems engineering*, we refer to error-avoidance and error-detection activities in control system and embedded software development (e.g. according to the V-model). Avizienis et al. [7] devised a comprehensive terminology and an overview of the assessment and handling of a variety of faults, errors, and failures. For critical systems, such activities are expected to be explicit (e.g. traceable, documented), to employ best practices (e.g. design patterns), and to be driven by reasonably qualified personnel (e.g. well-trained and experienced engineers or programmers).

The need for *dependability* often arises from the embedding of software into a cyber-physical context (i.e., an electronic execution platform, a physical process to be controlled, and other systems or human users to interact with). *Dependability assurance* (DA), or assurance for short, encompasses the usually cross-disciplinary task of providing evidence for an assurance case (e.g. safety, security, reliability) for a system in a specific operational context [104].

By *formal methods*, we refer to the use of formal (i.e., mathematically precise and unambiguous) modelling languages to describe system elements, such as software, hardware, and the environment, and the subjection of models written in these languages to analysis [101, 125], the results of which are targeted at assurance [31, 159]. FMs always require the use of both *formal syntax* and *formal semantics* (i.e., the mapping of syntax into a mathematical structure). Semantics that allows the verification of refinement or conformance across different FMs is said to be *unifying* [89, 168]. iFMs allow the coordinated application of several potentially heterogeneous FMs, supported by interrelated layers of formal semantics [14, 69].

FMs stand in contrast to *informal methods*, which employ artefacts without a formal syntax or semantics, such as natural language descriptions and requirements. In the gap between informal methods and FMs there is also a variety of *semi-formal methods*, including languages like the *Unified Modelling Language* (UML) and the *Systems Modelling Language* (SysML), whose syntax and semantics have frequently been subject of formalisation in research [e.g. 20, 54, 153, 175].

FM-based tools assist in the modelling and reasoning based on a FM. *Model-based development* (MBD) and *model-driven engineering* (MDE) served many opportunities for FM-based tools to be applied in dependable systems practice [12, 177].[4]

We speak of *applied or practical FMs* to signify successful applications of FMs in a practical context, for example, to develop embedded control software deployed in a commercial product marketed by an industrial company. We consider the use of FMs in research projects still as *FM research*. *Empirical FM research* investigates practical FMs, for example using surveys, case studies, or controlled field experiments [65]. We speak of *FM transfer* if FM research is transferred into practice with the aim to effectively apply and practice FMs. We consider FM transfer, as discussed below, as crucial for empirical FM research and progress of iFM research.

2.2 Related Work

Many researchers have suggested that FMs will, in one way or another, play a key role in mastering the difficulties discussed below and in achieving the desired guarantees (e.g. dependability, security, performance) of future critical systems [e.g. 101, 125].

Expecting an increased use of FMs to solve practical challenges in the mid 1990s, Clarke and Wing [31] suggested FM integration, tool development, and continuous specialist training to foster successful FM transfer to practice.

In 2000, van Lamsweerde [169] observed a growing number of FM success stories in requirements engineering. Evaluating several FM paradigms, he outlined weaknesses (e.g. isolation of languages, poor guidance) to be compensated and challenges to be met towards effective FM use, particularly their integration into multi-paradigm specification languages.

Aiming at the improvement of software dependability, Jackson et al. [96] made several key observations of recent dependability practice (e.g. lack of evidence for method effectiveness) leading to a general proposal with broad implications: rigorous dependability cases with explicit claims, the support of reuse and evolution, and the selective use of FMs. Additionally, these authors provide a number of recommendations to tool vendors and organisations in education and research.

Also in the mid 2000s, Hinchey et al. [84] spotted a decline of internet software dependability in the context of an increased level of concurrency in such systems. Their observation was backed by an earlier comparative software/hardware dependability discussion by Gray and Brewer [66]. Hinchey et al. highlighted achievements in FM automation enabling an increased use of lightweight FMs in "software engineers' usual development environments". Furthermore, they stressed the ability to use several FMs in a combined manner to verify distributed (embedded) systems, avoid errors and, hence, stop the decline of software dependability.

Hoare et al. [90] issued a manifesto for a "Verified Software Initiative". Based on a consensus of strengths, weaknesses, opportunities, and threats in the software engineering community, they proposed a long-term international "research program towards the construction of error-free software systems". This initiative aims to achieve its agenda through (1) new theoretical insights into software development, (2) creation of novel automated FM tools, and (3) a collection of experiments and benchmarks. In particular, the initiative is driven by a number of "grand challenges" [176]—difficult practical verification problems that can guide future research. The experiments have broad scope, and include a smart cash card [166] (the Mondex card), a secure entry system[5] (Tokeneer), and a cardiac pacemaker.[6]

Outlining an agenda for FM transfer, Jhala et al. [99] raised the need for improved benchmarks, metrics, and infrastructure for experimental evaluation, the need for revised teaching and training curricula [150], and the need for research communities interested in engaging with practitioners and working on ways to scale FMs up to large systems and to increase the usability of FMs. The authors specified several applications with great opportunities for FM transfer.

Applied researchers and practitioners interviewed by Schaffer and Voas [162] convey an optimistic picture of FM adoption in practice, highlighting the potentials to improve IT security, particularly in cyber-physical systems. Chong et al. [29] share the view that FMs are the most promising approach towards acceptably dependable and secure systems.

[4]The SCADE Design Verifier (http://www.esterel-technologies.com) and the seL4 microkernel (http://sel4.systems) represent good although less recent examples.

[5]Tokeneer Project Website: https://www.adacore.com/tokeneer.

[6]Pacemaker Formal Methods Challenge: http://www.cas.mcmaster.ca/sqrl/pacemaker.htm.

The challenges they list for the security domain are similar to the challenges we perceive in RAS assurance: FM integration, sound abstraction techniques, compositional guarantees, and evidence for sustainable transfer.

With their survey of FMs for RAS verification, Luckcuck et al. [122] identified difficulties of applying FMs in the robotics domain and summarised research results and their limitations. They conclude (i) that formalisation remains the most critical and most difficult task, (ii) that the surveyed approaches do not provide "sufficient evidence for public trust and certification", and (iii) that iFMs would be highly desirable if the current lack of translations between the most relevant of the surveyed techniques (e.g. model checking) could be overcome. We complement their observations with a further analysis of the lack of unification of FMs and of the missing empirical evidence for the effectiveness of FMs and iFMs. Additionally, we provide a research road map.

3 Strengths and Weaknesses of Formal Methods for Assurance

Following the guidelines for *strengths, weaknesses, opportunities, and threats* (SWOT) analysis by Piercy and Giles [152], we provide an overview of strengths and weaknesses of FMs, regarding

- reputation, proof culture, education, training, and use (Section 3.1),
- transfer efforts (Section 3.2),
- evidence of effectiveness (Section 3.3),
- expressivity (Section 3.4), and
- integration and coordination (Section 3.5).

3.1 Reputation, Proof Culture, Education, Training, and Use

In the guest editor's introduction of the "50 Years of Software Engineering" IEEE Software special theme issue [41], the question

> "Are formal methods essential, or even useful, or are they just an intellectual exercise that gets in the way of building real-world systems?"

invited us to deliberate on this topic and summarise its highlights. Applied researchers have raised the issue of *limited effectiveness and productivity* of FMs, particularly in large practical systems with changing requirements [55, 146]. FMs are known to be *difficult to apply in practice*, and *bad communication* between theorists and practitioners sustains the issue that FMs are taught but rarely applied [55]. In contrast, they are considered to have significant potential to cope with the toughest recent engineering problems: certifiable RAS assurance [45].

Studying the sociology of proof, MacKenzie [125] identified three sources of knowledge about a system's dependability: induction (i.e., from observation), authority (e.g. expert opinion), and deduction (i.e., inference from models) which is possibly the most powerful. Since the beginning of software engineering there has been a debate on the style of deductive reasoning about programs and on the usefulness of FMs. De Millo et al. [35] argued that proof is a social process. Long and difficult to read computer-produced verification evidence cannot be subject to such a process and is not genuine proof. Dijkstra [38] countered, albeit not as a supporter of mechanisation, to change from a personal trust-based culture of proof to the formalisation of proof steps. Fetzer [48] doubted that verification based on a model of the program can yield any knowledge about the dependability of an implementation of that program. According to Naur [139], it is not the degree of formalisation making a proof convincing but the way the argument is organised. MacKenzie tried to arbitrate this debate between rigorous proof in ordinary mathematics and formal mechanised proof. He suggested that proof assistants have the potential to use formal methods [101] to the maximum benefit. Daylight [34] concluded from a discussion with Tony Hoare that formalist and empiricist perspectives, while still causing controversies between research and practice, complement each other in a fruitful way.

Nevertheless, FMs have shown to be well-suited to *substantially improve modelling precision, requirements clarity, and verification confidence*. FM applications in requirements engineering such as the "Software Cost Reduction" tool set [83] even carry the hypothesis of FM cost-effectiveness in its name. By the 1990s, FM researchers had already started to examine FM usefulness with the aim to respond to critical observations of practitioners [9, 16, 73, 110, 120]. Some of these efforts culminated in empirical studies [151, 163] suggesting *high error detection effectiveness*, though with some controversy also caused by employed research designs [11, 164].

Jones and Bonsignour [100, Sec. 3.2, Tab. 3.2] observe that the combination of *formal*[7] *inspection*, static analysis, and formal testing has been the best approach to defect prevention with *up to 99% of accumulated defect removal efficiency*.

[7] "Formal" refers to the formality of the procedure but does not imply the use of formal semantics or mathematical techniques.

FMs can be seen as a rigorous and systematic form of this approach, though less often applied. In Appendix A, we make a brief excursion to the relationship between FMs and formal inspection and try to roughly estimate the population size of FM users.

From two larger surveys, one in the early 1990s [6] and another one in the late 2000s [12, 177], we obtain a more comprehensive picture of the typical advantages of FM use and barriers to FM adoption as seen by practitioners and practical FM researchers. In two recent surveys [59, 60], we made two, not necessarily surprising but empirically supported, observations underpinning the main findings of the former studies: many practitioners *view FMs as promising* instruments with high potential, and would use these instruments to their maximum benefit, whether directly or through FM-based tools. However, the beneficial use of FMs is still hindered by severe obstacles (e.g. FMs are considered hard to learn, difficult to integrate in existing processes, too expensive, prone to invalid abstractions, difficult to maintain).

Strength 1 *FMs can improve RAS modelling, the specification of RAS requirements, and the automation of RAS verification, fostering the early detection of systematic errors in RAS designs. Many assurance practitioners perceive FM usefulness as positive.*

Weakness 1 *FMs have shown to be difficult to learn and apply. Many assurance practitioners perceive the ease of use of FMs as negative. Moreover, research has been ineffectively communicated in FM teaching and training.*

3.2 Transfer Efforts

FMs can be effective in two ways, *ab-initio* (i.e., before implementation) and *post-facto* (i.e., after implementation). The ab-initio use of FMs aims at reducing late error costs through, for example, formal prototyping, step-wise refinement, formal test-driven development, crafting module assertions prior to programming, or formal contract-based development. Once an initial formalisation (e.g. invariants) is available, it is argued for families of similar systems that, from the second or third FM application onward, the benefit of having the formalisation outperforms the cumulative effort to maintain the formalisation up to an order of magnitude [96, 129, 130]. This argument also addresses agile settings inasmuch as iterations or increments refer to similar systems. The post-facto use of FMs can occur through knowledge extraction from existing artefacts and using automated tools such as, for example, formal or model-based post-facto testing tools or post-facto use of code assertion checkers [103, 118]. Overall, the second way of utilising FMs is known to be more compatible with everyday software practice.

Achievements collected by Aichernig and Maibaum [2], Boulanger [15], and Gnesi and Margaria [62] show that many researchers have been working *towards successful FM transfer*. Moreover, researchers experienced in particular FMs draw positive conclusions from FM applications, especially in scaling FMs through adequate tool support for continuous reasoning in agile software development [131, 142]. Other researchers report about progress in theorem proving of system software of industrial size [e.g. 107] and about FM-based tools for practical use [e.g. 12, 103, 149]. MBD and MDE have a history of wrapping FMs into software tools to make access to formalisms easier and to help automating tedious tasks via *domain-specific language*s (DSLs) and visual notations.

Static (program) analysis is another branch where FM-based tools have been successfully practised [e.g. 103]. However, few static analysis tools are based on FMs and many of these tools are exposed to reduced effectiveness because of *high false-positive rates*, particularly if settings are not perfectly adjusted to the corresponding project [58].

Furthermore, the concolic testing technique [63], a post-facto FM, has seen multiple successes in industry [64, 105]. It exercises all possible execution paths of a program through systematic permutation of a sequence of branch conditions inferred by an instrumented concrete execution. It uses these symbolic execution paths and *Satisfiability Modulo Theory* (SMT) solving to obtain a series of inputs that exercise the full range of program paths. It does not depend on a predefined model of the program, but effectively infers one based on the branch conditions. It can therefore readily be used on existing program developments, and has notably been used by Samsung for verification of their flash storage platform software [105]. Indeed, it is a belief of the authors of this latter work that post-facto methods provide greater opportunities for adoption of FMs in industry.

Recently, there have been several developments in the use of FMs to assure safety requirements related to *human-computer interaction* (HCI), and a growing body of literature [76, 144, 173], particularly relating to certification of commercial medical devices [17, 126]. There, formal methods have been shown to be successful in facilitating regulatory acceptance, for example with the *Food and Drug Administration* (FDA) agency in the United States. Masci et al. [126] formalised the FDA regulatory safety requirements for the user interface of a patient-controlled infusion pump, and used the PVS proof assistant to verify it. Bowen and Reeves [17], similarly, formally modelled an infusion pump interface, using the Z notation, and then used the resulting specification as an oracle to generate test cases. Harrison et al. [75] performed risk analysis for a new neonatal dialysis machine, formalised safety requirements, and use these requirements to analyse and verify the source code. The consensus of these authors is that FMs can greatly reduce the defects in safety critical HCI prior to deployment. The techniques are increasingly practically applicable through greater automation. Moreover, it is likely that these results have potential applications in other domains, such as RAS, provided they can be transferred and scale to this greater complexity.

Strength 2 *There exist many transfer re-entry points from a range of insightful FM case studies in industrial and academic labs. FMs were demonstrated to be a useful basis for many static analysis and MDE tools.*

Weakness 2 *The number of practical (comparative) case studies using (ab-initio) FMs or iFMs, particularly on RASs, is still too low to draw useful and firm conclusions on FM effectiveness.*

3.3 Evidence of Effectiveness

Whether used as ab-initio or post-facto tools, *strong evidence* for the efficacy of FMs in practice is *still scarce* [e.g. 151] and more anecdotal [e.g. 2, 15, 62, 162], rarely drawn from comparative studies [e.g. 151, 163], often primarily conducted in research labs [e.g. 30, 52], or not recent enough to reflect latest achievements in verification tool research [e.g. 28]. We observe that a large fraction of empirical evidence for FM effectiveness can be classified as level 6 or 7 according to [65, Tab. 2], that is, too weak to draw effective conclusions.

Jackson et al. [96, p. 39] as well as Jones and Bonsignour [100, Sec. 4.4, p. 220], two researchers from the software engineering measurement community, support this observation. Jones and Bonsignour state that "there is very little empirical data on several topics that need to be well understood if proofs of correctness are to become useful tools for professional software development as opposed to academic experiments". Moreover, the controversies about proof culture summarised in Section 3.1 contain little data to resolve practitioners' doubts.

Graydon [67] observed this lack of evidence of FM effectiveness in assurance argumentation. More generally, Rae et al. [154] noticed insufficiently evaluated safety research. About 86% of works lack guidance to reproduce results, hence forming a barrier to the advancement of safety practice. Although their study is limited to one conference series, it indicates deficiencies in the evaluation of DA research. Overall, it is important to understand that the mentioned lack of evidence and successful transfer produces great opportunities for further empirical and theoretical FM research.

Strength 3 *For (comparative) studies of FM effectiveness, there are several research designs and benchmark examples available from the scientific literature. In Appendix A, we assess the effort and feasibility of corresponding qualitative and quantitative studies.*

Weakness 3 *FMs have been suffering from fragile effectiveness and productivity in dependability engineering in general. There is a lack of convincing evidence of FM effectiveness, particularly, of ab-initio FMs. RAS engineering and assurance are likely to be affected by these weaknesses.*

3.4 Expressivity

An often quoted weakness of MBD, particularly when applied to RASs, is the "reality gap" [21, 97] that can exist between a naively constructed model and its corresponding real-world artefact. According to Brooks [21], over-reliance on simulation to test behaviour using naive and insufficiently validated models can lead to effort being applied to solving problems that do not exist in the real world. Worse, programs for robotic controllers developed in a model-based setting may fail when executed on real-world hardware, because "it is very hard to simulate the actual dynamics of the real-world" [21]. This problem is not only true of simulation, but any form of model-based analysis, including reasoning in FMs [48].

The fundamental problem here is that it is impossible to model the behaviour of any physical entity precisely [117], unless we replicate the original. Moreover, as models become more detailed, their utility decreases and they can become just as difficult to comprehend and analyse as their real-world counterparts, an observation highlighted by the famous paradox of Bonini [13]. Nevertheless, as statistician George Box said "all models are wrong but some are useful" [19]: we must evaluate a model not upon how "correct" it is, or how much detail it contains, but on how *informative* it is. According to Lee and Sirjani [117], the antidote is not to abandon the use of models, but to recognise their inherent limitations and strengths, and apply them intelligently to reasoning about a specific problem. This means selecting appropriate modelling paradigms that enable specification of behaviour at a sufficiently detailed level of abstraction, and using the resulting models to guide the engineering process.

Strength 4 *FMs allow and foster the use of specific abstractions to specifically inform engineers of RAS properties critical for their assurance.*

Weakness 4 *The effectiveness of formal models is fragile and can be significantly reduced because of uncontrollable gaps between models and their implementations.*

3.5 Integration and Coordination

Modelling notations usually employ a particular paradigm to abstract the behaviour of the real-world. For example, the state-based paradigm, employed by FMs like Z [165], B [1], and refinement calculus [8, 136], considers how the internal state of a system evolves, whilst the event-driven paradigm, employed in process calculi like CSP [86], CCS [132], and π-calculus [133], considers how behaviour may be influenced by external interactions. Consequently, individual formal methods are usually limited to considering only certain aspects or views of a system's behaviour [26, 145], which can limit their effectiveness when used in isolation. Many researchers have therefore sought to overcome this weakness by FM integration [26, 31, 53, 145].

The 1990s saw a large number of works on semantic unification and method integration [53, 145]. Theoretical foundations were provided by Hehner, in his seminal work on semantic unification using the "programs-as-predicates" approach [79, 80] and comparative semantics [82]. At the same time, refinement calculi were developed [8, 136, 137] that would underlie the work on linking heterogeneous notations through abstraction. Also, Woodcock and Morgan [178] explored the integration of state- and event-based modelling using weakest preconditions, and several other works on this topic followed [42, 51, 157]. Hoare proposed a unified theory of programming [87] that links together the three semantic styles: denotational, operational, and algebraic. These developments culminated in Hoare and He's *Unifying Theories of Programming* (UTP) [89], a general framework for integration of semantically heterogeneous notations by application of Hehner and Hoare's approach [81] to the formalisation of a catalogue of computational paradigms, with links between them formalised using Galois connections. This framework enables a definitive solution to the integration of states and events, along with other computational paradigms, in the CIRCUS language family [27, 143, 171].

Another result of these developments was a number of seminal works on FM integration [26, 53, 145]. Paige, inspired by work on systematic method integration [113], defined a generic "meta-method" that aimed at integration of several formal and semi-formal methods using notational translations with a common predicative semantic foundation, which builds on Hehner's work [80]. Meanwhile, Galloway and Stoddart [53], building on their previous work [51], likewise proposed the creation of hybrid FMs with a multi-paradigm approach. Moreover, Broy and Slotosch [26] proposed that FMs should be integrated into the V-Model of development [92] with common semantic foundations to link the various artefacts across development steps.

These diverse efforts eventually led to the founding of the iFM conference series in 1999 [4], with the aim of developing theoretical foundations for "combining behavioural and state-based formalisms". For the second iteration of the iFM conference [69], the scope broadened to consider all the different aspects of FM integration, including semantic integration, traceability, tool integration, and refinement. A few years later, a conference series was also established for UTP [39], with the aim of continuing to develop unifying semantics for diverse notations within the UTP framework.

However, there is as yet no agreed and general methodology for integrating FMs that could be applied to RASs [45]. Overall, integration is of particular pertinence to RASs, since such systems are multi-layered and possess a high degree of semantic heterogeneity. As Farrell et al. found, they "can be variously categorised as embedded, cyber-physical, real-time, hybrid, adaptive and even autonomous systems, with a typical robotic system being likely to contain all of these aspects". When we consider RASs, we must consider advanced computational paradigms like real-time, hybrid computation with differential equations, probability, and rigid body dynamics. This implies the use of several different modelling languages and paradigms to describe the different aspects, and therefore a variety of analysis techniques to assure properties of the overall system. Assurance of autonomous systems will certainly therefore require iFMs [45]. Figure 1 on page 3 summarises this relationship.

Strength 5 *iFMs raise the potential of integration and coordination of several FMs to consistently reason about RAS properties implemented by a combination of various technologies such as software, electronic hardware, and mechanical hardware.*

Weakness 5 *There is currently no agreed framework for the integration of FMs that would effectively address the needs in the RAS domain or similar domains.*

4 Opportunities for Integrated Formal Methods

This section continues with the *environmental part* of our SWOT analysis. Several key opportunities for the transfer of iFMs arise from ongoing assurance challenges, particularly in RAS assurance and from looking at what other disciplines do to cope with similar challenges. In the following, we describe opportunities stemming from

- the desire for early removal of severe errors (Section 4.1),
- the desire to learn from accidents and their root causes (Section 4.2),
- the desire of assurance to form a mature discipline (Section 4.3), and
- the desire for adequate and dependable norms (Section 4.4).

4.1 The Desire for Early Removal of Severe Errors

Summarising major challenges in automotive systems engineering in 2006, Broy [22, p. 39] indicated that modelling languages used in practice were often not formalised and the *desired benefits could not be achieved from semi-formal languages*. Moreover, *software engineering was not well integrated* with core control and mechanical engineering processes. Domain engineers would produce software/hardware sub-systems and mechanical sub-assemblies in undesirable isolation. Broy referred to a lack of iFMs for *overall architecture verification*.

Has the situation changed since then? In model-centric development in embedded software practice (e.g. based on UML or SysML), drawbacks can be significant if methods and tools are not well integrated or trained personnel are missing [119]. Likely, Broy's criticism remains in contemporary automatic vehicle engineering and assurance practice. In fact, he has a recent, clearly negative, but not pessimistic answer to this question [24]. Moreover, this view is shared by the Autonomy Assurance International Programme's discussion of assurance barriers,[8] that is, current challenges in RAS assurance. These barriers (e.g. validation, verification, risk acceptance, simulation, human-robot interaction) could be addressed by formal engineering models and calculations based on such models to be used as evidence in assurance cases.

Model-based assurance [72, 77] uses system models to structure assurance cases and represents another opportunity for formal methods in (through-life) assurance. Assurance arguments that are purely informal can be difficult to evaluate, and may be subject to argumentation fallacies [68]. Consequently, there have been a number of efforts to formalise and mechanise assurance cases, both at the argumentation level [37, 161] and the evidence level [32]. More recently, in MDE, the *Structured Assurance Case Meta-model* (SACM)[9] is a standardised meta-model that supports both structured argumentation and integration of evidence from diverse system models [172]. SACM unifies several existing argumentation notations, and also provides support for artefact traceability and terminology. It could, in the future, serve as a crucial component for iFMs in assurance.

Leading voices from applied software engineering research periodically mention the role of FMs as a key technology to master upcoming challenges in assuring critical software systems [135]. A round table about the adoption of FMs in IT security [162] positively evaluated their overall suitability, the combination of FMs with testing, and the achievements in FM automation. The panellists noticed limitations of FMs in short-time-to-market projects and in detecting unknown vulnerabilities as well as shortcomings in FM training and adoption in practice.

However, even for mission-critical systems, high costs from late defect removal and long defect repair cycles [100], as well as dangerous and fatal[10] incidents indicate that assurance in some areas is still driven by practices failing to assist engineers in overcoming their challenges. Moreover, Neumann, an observer of a multitude of computing risks, stated that "the needs for better safety, reliability, security, privacy, and system integrity that I highlighted 24 years ago in my book, Computer-Related Risks, are still with us in one form or another today" [47, 91, 140].

For example, artificial intelligence software—particularly *machine learning* (ML) components—has been developed at a high pace and used in many non-critical applications. Recently, ML components are increasingly deployed in critical domains. For verification and error removal, such software has to be transparent and explainable. Preferring verifiable algorithms to heuristics, Parnas [147] recalled the corresponding engineering principle: "We cannot trust a device unless we know how it works". One way to follow this principle and establish transparency is to reverse engineer (i.e., to decode) the functionality of an ML component even if this is not possible in general [10]. FMs can

[8]See https://www.york.ac.uk/assuring-autonomy/body-of-knowledge/.

[9]See https://www.omg.org/spec/SACM/About-SACM/.

[10]For example, the fatal accident involving a Tesla advanced driving assistance system, https://www.theguardian.com/technology/2018/jun/07/tesla-fatal-crash-silicon-valley-autopilot-mode-report.

help extract knowledge and reverse engineer abstractions of ML systems to explain their behaviour. We might then ask to which extent the reverse engineered and verified functionality serves as a substitute for the original ML component.

These anecdotes make it reasonable to question current assurance practice. Seen through the eyes of assurance, they suggest that we might again be facing a dependable software engineering *crisis* similar to the one from the late 1960s [24, 156].

Opportunity 1 *We as method researchers could learn from this crisis and improve the way that FMs can be effectively coordinated to support early error removal in practical engineering processes. We as practitioners could learn from this crisis and improve the way that we correctly engineer and certify highly automated systems such as robots and autonomous systems.*

4.2 The Desire to Learn From Accidents and Their Root Causes

In the title of Section 4.1, the word “severe” refers to the negative consequences *potentially* caused by errors we want to remove using iFMs. The more severe the potential consequences of an error, the more critical is its early removal. The usefulness of iFMs thus positively correlates with their support in the removal of critical errors. However, the estimation of severity often also requires the careful study of past field incidents [93].

We speak of *field incidents* to refer to significant operational events in the *field* (i.e., the environment a technical system is operated) which are undesired because of their safety risks and their severe harmful consequences. Field incidents range from minor incidents to major accidents. It is important to separate the observed effect, the field incident, from its causes or, more precisely, from the *causal chains of events* leading to the observed effect. Hence, this analysis depends on the considered system perimeter [e.g. 7]. Depending on the possibilities of *observation* and the depth pursued in a *root cause analysis* (RCA), a conclusion on a possible cause can result in any combination of, for example, overall system failure, human error, adverse environmental condition, design fault, hardware fault, software fault, or specification error.[11]

There are many databases about field incidents: some are comprehensive and include RCA, others are less detailed, and some are confidential, depending on the regulations in the corresponding application domain or industry sector. Based on such databases, accident research, insurance, and consumer institutions occasionally provide brief root cause statistics along with accident statistics.[12]

Accident statistics allow certain predictions of the safety of systems and their operation, for example whether risk has been and will be acceptably low. Such statistics are also used in estimations of the amount of field testing necessary[13] to sufficiently reduce risk [102].

For example, it is well-acknowledged that inadequately designed and implemented user interfaces are a significant contributory factor in computer-related accidents [112, 114, 124]. But how and how fast have we arrived at this conclusion and how can we prevent future such incidents? Without proper empirical investigation of accident causes, such statistics are *of little use in decisions on measures for accident prevention* [94], particularly on improvements of engineering processes, methods (e.g. iFMs), and technologies (e.g. iFM tools) used to build these systems. For this, we require more in-depth, possibly formal, RCAs and statistics that *relate error removal by iFMs and incident root causes*. To this extent, RCA is a great opportunity for the investigation of iFM effectiveness.

MacKenzie [124] reported about deficiencies of information gathering (e.g. RCA) for databases on computer-related accidents back in the early 1990s. He noted that independent, well-known, but confidential databases might reduce under-reporting and were thus believed to improve safety culture. Reporting has not much improved as observed by Jackson et al. [96, p. 39]. To understand the current RCA situation, we studied a sample of 377 reports from open field incident databases (in aviation, automotive, rail, energy, and others) finding the following [60]:

1. RCAs in these reports were of poor quality, either because they were not going deep enough, economically or technically infeasible, or inaccessible to us.
2. Particularly, root causes (e.g. software faults, specification errors) were rarely documented in a way that useful information about the technologies used (e.g. software) or consequences in the development process could be retrieved from the reports.

[11]Specification errors are also called development failures [7] and can be seen as flaws in the process of requirements validation.

[12]See [56, Sec. 1.1] for a list of accident databases from such institutions.

[13]For example, according to “As Low As Reasonably Practicable” or “So Far As Is Reasonably Practicable”. See http://www.hse.gov.uk/risk/theory/alarpcheck.htm: “Something is reasonably practicable unless its costs are grossly disproportionate to the benefits.”

3. Reports in some sectors (e.g. aerospace, rail, power plants, process industry) contain more in-depth RCAs than others (e.g. automotive) because of different regulations.
4. Some sectors operate official databases (e.g. NHTSA[14] and NTSB[15] in the US transportation sector) and others do not (e.g. German road transportation sector).
5. Our findings suggest that even in domains with regulated RCA, reports in open databases tend to be less informative than reports in closed databases.
6. The reports from the automotive industry exhibited a relatively small fraction of technology-related errors (e.g. software-related errors).

To validate our study and to better understand the context of our findings, we interviewed eight[16] safety experts [60]. One finding was that, because of an unclear separation of technologies and a lack of explicit architectural knowledge, a desirable classification of root causes is sometimes infeasible. Hence, accident analysts often close their reports with a level of detail too low to draw helpful conclusions. Additionally, one expert stated that *the hidden number of software-related or software-caused field incidents in dependable systems practice is likely much larger than the known number*. This matches our intuition but we are missing clear evidence.

Ladkin, a researcher involved in the further development of IEC 61508, demands regulations to mandate the use of systematic RCAs.[17] In support of his view, we believe that rigorous in-depth RCAs based on iFMs can be helpful to gain clarity about actual root causes. Again, beyond this undesirable form of late error removal, RCA data is essential for the *measurement of the effectiveness* of error removal techniques, particularly iFMs.

The "Toyota unintended acceleration" incident exemplifies the difficulty of drawing conclusions without using powerful RCA techniques: a first RCA concluded that floor mats and sticky throttle pedals caused a fatal car mishap. A second RCA carried out by NASA experts and based on *testing and automated static analysis* of the control system (i.e., software and hardware) was not conclusive. A third RCA[18] based on *code reviews*—we could not find out which level of formal inspection was used—detected defects in the control software and safety architecture, demonstrated to be likely the causes of the accident [111].

Opportunity 2 *We could invest in integrated formal methods for RCA based on standardised data recording (e.g. aircraft black boxes), especially important for RASs where human operators cannot practically perform incident response. Based on lessons from formal RCA, we could further invest in the employment of integrated formal methods in RAS assurance and certification to prevent field incidents, major product recalls, and overly lengthy root cause investigations.*

4.3 The Desire of Assurance to Form a Mature Discipline

In his Turing Award acceptance speech in 1981, Tony Hoare reviewed type safety precautions in programming languages and concluded: "In any respectable branch of engineering, failure to observe such elementary precautions would have long been against the law" [85].

Inspired by this comparison, it can be helpful to look at other engineering disciplines such as civil, mechanical, or electrical engineering to identify transfer opportunities for iFMs. There, engineers use FMs in many of their critical tasks. However, nowadays these methods are often hidden behind powerful software tools usable by qualified professional engineers. Although type systems, run-time bounds checking, and other variants of assertion checking have been frequently used in dependable systems practice, the overall level of FM adoption is still comparatively low.

For example, even in less critical mechanical engineering domains, vocationally trained engineers use computer-aided engineering, design, and manufacturing software. Whether for designing machine parts for serial production (i.e., specification) or for calculations (e.g. dimensioning, force or material flow simulations) for these parts and their assembly (i.e., for prototype verification), these engineers use tools based on canonical mathematical models.

Nowadays, drawings from computer-aided mechanical design carry at least two types of semantics: one declarative based on calculus for dimensioning (1), and one procedural for the synthesis of Computer-Numerical-Control programs for production machines processing materials to realise the drawings (2). Note that the unifying base of these

[14]National Highway Traffic Safety Administration, https://www.nhtsa.gov.

[15]National Transportation Safety Board, https://www.ntsb.gov.

[16]These experts have experience with tool qualification (1) and safety-critical systems in aviation (3), railway (1), automotive (4), and energy systems & turbines (1). We have to keep their names confidential.

[17]From personal communication.

[18]See expert interview by embedded software journalist from EE Times in 2013 on https://www.eetimes.com/document.asp?doc_id=1319903&page_number=1.

two semantics is geometry, a well-studied mathematical discipline. Although higher levels of complexity demand more sophisticated analytical expertise, typically from engineers with several years of work experience, many tasks can be accomplished by less trained engineers using the corresponding tools.

Whereas in computer-aided mechanical design both semantics seem to be used to a similar extent, in DA we observe that analogous semantics are rarely used even if tools are available, and less often we see (1) and (2) being consistently used. Low adoption might result from the semantics for dimensioning and production automation being usually less abstract than the semantics for verification (1) and synthesis (2) of computer programs. Accordingly, Parnas suggests a shift from correctness proof to property calculation to develop practical formal methods [146, p. 33].

Patterns have had a long history in many disciplines. In mechanical engineering, patterns are better known as *machine elements* and are particularly useful in high-reliability applications. Machine elements (and standardised forms thereof) have a stabilising impact on the outcome of an engineering project. The process of element selection and composition can take tremendous advantage not only from the *reuse* of proven design knowledge but also from the reuse of complex calculations (e.g. from gear transmissions, injection moulding tools, skeleton framings). Moreover, modern tools typically foster the use of *element libraries* and *parametric design*. Importantly, because the properties of such elements are in many cases well known, calculations for assemblies (i.e., *compositional* verification) get relatively easy. However, the higher the required precision of these calculations, the more expensive is their computation.

These observations are in line with what we know from collaborations in robotics, like mechatronics, a discipline where many engineering domains have to play together well: FMs are heavily used for the analysis of robot controllers and for various kinds of simulations and tests [121, 128].

Digital circuit engineering is a domain where FMs such as model checking have been successfully applied decades ago. However, systematic hardware errors, such as Spectre and Meltdown, and the *unavailability of temporal specifications* of optimised operations (e.g. branch-prediction and speculative execution) discontinue the verifiability of recent computer architectures. This lack of verifiability of the assumptions (e.g. partitioning, information flow) about the execution platform complicates the verifiability of the software (e.g. an operating system) running on such a platform.[19]

Opportunity 3 *Dependability assurance has not yet successfully adopted iFMs as a vital part of their key methodologies. If FMs seem relatively well established in other disciplines, we might also be able to successfully transfer iFMs to RAS assurance and assurance in other domains. Beyond software design patterns, we could benefit from best practices in formal specification and development manifested in repositories of FM patterns. Moreover, we could aim at the further unification of established FMs to provide common formal semantics for various domains.*

4.4 The Desire for Adequate and Dependable Norms

Dependable systems practice usually includes the transition from what is called the *system specification* to an artefact called *system design*. Typically, *software and hardware specifications* are then derived from these two artefacts before delving into detailed technology choice and development. Jackson et al. [96, p. 48] observe that safety culture in such a framework is more important than strict standards, but adequate standards and certification regimes can establish and strengthen the safety culture desirable in dependable systems practice.

A striking finding in one of our recent discussions of dependable systems standards (e.g. IEC 61508, ISO 26262,[20] DO-178C) is that *normative parts for specification* (i.e., requirements engineering), for specification validation (i.e., avoiding and handling requirements errors), *and for hazard and risk analysis* (particularly in early process stages) seem to be *below the state of the art* [47, 60], despite several observations that significant portions [e.g. 44%, 78] of the *causes* of safety-critical software-related incidents fall into the category of *specification errors* [108, 109].

If one identifies an error in the software specification, do causes of this error originate from an erroneous system specification? How are system, software, and hardware specifications (formally) related? Control software typically governs the behaviour of a whole machine. Hence, core components of a software specification might, modulo an input/output relation [148], very well govern the narrative of the overall system specification. Inspired by the issue of specification validity as highlighted in [101], do these standards provide guidance for checking the validity and consistency of core parts of system, software, and hardware specifications?

The literature provides plenty of evidence of undesired impacts of specification errors dating back as early as the investigations of Lutz [123] and Endres [40]. As reported by MacKenzie [124], the 92% of computer-related field

[19]See blog post on the seL4 microkernel, https://research.csiro.au/tsblog/crisis-security-vs-performance/.
[20]For: Road Vehicles – Functional Safety.

incidents caused by human-computer interaction also illustrate the gap between specifications and capabilities of humans to interact with automation. Despite these older figures, we are talking of one of the most critical parts of standards. Practitioners could expect to receive strong guidance from these parts. Moreover, the requirements to show conformance to these parts should not be vacuous.

Many standards define *specific sets of requirements* (i.e., for error removal and fault-tolerance) depending on the level of risk a system (or any part of it) might cause. The higher this level, the more demanding these requirements. Examples of demanding requirements include *safety integrity level* (SIL) 3-4 (IEC 61508), *automotive safety integrity level* (ASIL) C-D (ISO 26262), systematic capability 3-4 (IEC 61508), *design assurance level* (DAL) A-B (DO-178C). Even for the highest such levels the mentioned standards only "highly recommend" but not mandate the use of FMs.

Guidelines for embedded software development such as MISRA:1994 [138] *recommend* FMs for SIL 4, although MISRA:2004 no longer includes such information and instead refers back[21] to MISRA:1994. As already mentioned, ISO 26262 as the overriding standard does not go beyond high recommendation of FMs for ASIL D. Koopman [111] reported in 2014 that, in the US, car manufacturers are not required to follow MISRA and that there are no other software certification requirements. This currently also applies to autonomous road vehicles.

In an interesting anecdote, Ladkin reported on his lack of success in introducing systematic hazard (and risk) analysis methodology into normative parts of this standard [115]. Moreover, he mentioned[22] unsuccessful attempts to strengthen the role of FMs in IEC 61508 and on the "broken standardisation" in assurance practice. In reaction to that, he proposed the use of evidently independent peer reviews to "dampen committee-capture by big-company bully players".

Additionally, Knight [109] observed: "There is an expectation by the community that standards will embody the best available technology and that their presentation will allow determination of conformance to be fairly straightforward. A criticism that is seldom heard is that some standards are, in fact, technically flawed and poorly presented." He exemplifies his critique by several issues with IEC 61508 and RTCA DO-178B and suggests to make the meaning of "conformance [or compliance] with a standard" more rigorous. Particularly, he encourages to replace *indirect* (i.e., process-related) evidence (e.g. documentation of specification activities) in assurance cases by *direct* (i.e., artefact-related) evidence (e.g. unsuccessful checks for presence of certain specification faults, successful checks for absence of implementation errors).[23] With the observation in software quality control that "there is little evidence that conformance to process standards guarantees good products", Kitchenham and Pfleeger [106] delivered a reasonable basis for Knight's suggestions.

Regarding the integration of dependability approaches and FMs, Bowen and Stavridou [18] had stated by 1993 that they "do not know how to combine formal methods assurance with metrics collected from other techniques such as fault-tolerance". Is this still an issue? From a practical viewpoint, standards such as, for example, IEC 61508, ISO 26262, and DO-178C, provide recommendations about techniques for the reduction of both random hardware failures (e.g. by fault-tolerance techniques) and systematic hardware and software failures (e.g. by FMs, static analysis, and testing). If iFMs can support the combined application of the recommended techniques and achieve an improvement in practice then we should really strive to demonstrate this.

We believe that critical fractions of strong direct evidence can be delivered through the use of FMs. In support of Feitelson's argument [47], we see a great opportunity for an assessment of how the corresponding guidelines in these standards can be extended and aligned with recent results in FM research.

Opportunity 4 *We as researchers and practitioners could support the creation of adequate state-of-the-art regulations with improved guidance on specification construction and validation. That way, we could foster well-certified high-risk software in a time where dangerous autonomous machines are about to get widely deployed in our society.*

5 Threats to the Adoption of Integrated Formal Methods

This section closes the *environmental part* of our SWOT analysis by identifying threats to FM transfer as well as challenges that arise from alternative or competing approaches taking the opportunities mentioned in Section 4. We also outline remedies to these threats.

[21]This might also the case for MISRA:2012 from March 2013. We are unaware of the opposite but were also unable to receive a copy of this version.

[22]See System Safety Mailing List message from 4/11/2018, http://www.systemsafetylist.org/4183.htm and [116].

[23]While formal verification serves the check of absence of property violations, conventional testing can only serve as a check of presence of such violations.

The development of effective iFMs and their successful transfer into practice can be impeded by

- a lack of agreement on a sound semantic base for domain-specific and cross-domain FM integration (Sections 5.1 and 5.2),
- missing support for widely used and established tools (Section 5.2),
- a lack of interest in practical problems on the side of FM researchers (Section 5.3),
- a lack of incentives for FM researchers to engage with current practice and for software practitioners to engage with recent theoretical results (Section 5.3),
- a bad reputation among practitioners and applied researchers (Section 5.3),
- proofs that are faulty or do not scale (Section 5.4),
- the quest for soundness overriding the quest for usefulness (Section 5.5).

We discuss these threats and barriers in more detail in the following.

5.1 Difficulties and Misconceptions of Unification

According to Broy [22], the successes and failures of semi-formal languages (e.g. UML, SysML) suggest that FMs, once wrapped in FM-based tools, get exposed to the *quest for a unified syntax*, one main objective of the UML movement in the 1990s. Rather than a unified syntax, it is more desirable to have a unified semantics and several well-defined mappings to domain-specific syntax wherever convenient (Section 3.5). This approach is occasionally taken up by DSLs in MDE (Section 3.2). Harel and Rumpe [74] argued that one cannot achieve proper integration of methods and notations without a unifying semantics. This argument carries over to the problem of tool integration as already discussed in Section 3.5 and revisited below. Particularly, the following challenges apply to FMs when used in MDE:

1. the maintenance of a single source of information serving in the (automated) derivation of downstream artefacts (e.g. proof results, code via synthesis) [134],
2. a clear mapping between the DSL presented to the engineer (using intuitive notation) and the DSL semantics serving as the basis of formal verification,
3. the embedding of a lean domain-specific formalism into a *common* data model [25] suitable for access and manipulation by engineers through their *various* tools [61].

These challenges are complicated by irreducible unidirectionalities in automated transformations (e.g. model-to-code) limiting the desirable round-trip engineering [167] (i.e., the change between views of the same data).

We discussed SACM [172] as an assurance DSL in Section 4.1. Likewise, *architecture description languages* (e.g. the Architecture Analysis & Design Language [46], EAST-ADL [36]) are DSLs for overall embedded system design. DSLs can be seen as one shortcut to the still ongoing efforts of arriving at a reduced version or a variant of UML where a semantics can be defined for the whole language [e.g. 153].

At a higher level of abstraction, so-called *architecture frameworks* (cf. ISO 42010, e.g. the Department of Defense Architecture Framework) and *artefact and traceability models* [e.g. 155, 174] have been proposed, aiming at the standardisation of specific parts of the systems and software engineering life-cycle and of the documentation and data models used there. These frameworks and models are similar to the models used in product data/life-cycle management in fields like mechanical or civil engineering.

To our best knowledge, no cross-disciplinary semantic unification has been undertaken yet (see Section 3.5), serving as a basis for dependable systems engineering. Although many of these approaches have not been developed with the aim of formalisation and the unification of semantics, we believe that this effort has to be made when developing powerful iFMs.

Threat 1 *The main threat is the lack of agreement on a sound semantic base for domain-specific and cross-domain iFMs.*

Remedy 1 *To reduce this threat, FM integration and refinement-based software engineering could be better aligned with artefact models [e.g. 127, 174]. This alignment may foster the unification of formal semantics to strengthen traceability among the artefacts and to aid in a variety of change impact analysis tasks in the engineering process [e.g. 23, 170].*

5.2 Reluctant Integration Culture and Legacy Processes

Tool integration is about the integration of engineering information technology, for example, tools for requirements specification, computer-aided software engineering, and computer-aided mechanical design. Among the wide variety of solutions to capture and track model data, the majority deals with linking or merging data models [61] in one or another shallow way (e.g. software repositories, data exchange formats, product/engineering/application data or life-cycle management systems).

Some tools with sustainable support are heavyweight, making it difficult to agree on lean model semantics, while others are proprietary, accompanied with interest in hiding model semantics. The surveys of Liebel et al. [119, pp. 102,104], Mohagheghi et al. [134, p. 104], and Akdur et al. [3] confirm that method and model integration have not yet been solved in MBD, MDE, and dependable systems practice. Moreover, frequent proposals by researchers [e.g. 20, 54, 153] to formalise fragments or variants of UML and SysML have not yet received wide attention by practitioners and standardisation authorities.

DSL-based integrated development environments (e.g. using Xtext and Sirius) get close to what is suitable for FM-based tools. Such tools rely on a trusted representation of the formal semantics integrating the model data. For successful iFM transfer to assurance practice, tools need to be built on a lean and open central system model [e.g. 5, 95].

An even greater barrier than loosely integrated tools are legacy *language and modelling paradigms*, an *established tool and method market* carried by *legacy stakeholders* and, possibly, a *neglected continuous improvement of FM education and training*.

Threat 2 *The main threat is discontinuous and disintegrated FM education, transfer, and tool development.*

Remedy 2 *To reduce this threat, continuous adaptation and improvement of education through teaching, of transfer through training, application, and feedback, and of tool development through regulated interface standards are necessary.*

5.3 Reluctant Transfer Culture and Exaggerated Scepticism

Finally, the vision of introducing iFMs into assurance practice might be hindered by a lack of FM researchers able or willing to engage with industrial assurance practice, as diagnosed by Woodcock et al. [177]. It is certainly hard work to collect sufficient evidence for FM effectiveness in assurance practice because of intellectual property rights and other legal issues but also because of a lack of awareness among FM researchers [177]. However, for credible method comparison experiments, Jones and Bonsignour [100] recommended a sample of 20 similar projects split into two groups, 10 projects without treatment (i.e., not using FMs) and 10 projects with treatment (i.e., using FMs) to establish strong evidence (i.e., evidence of level 5 or above [65]).

Exaggerated scepticism on the side of practitioners and applied researchers that has piled up over the years might be the most important barrier to cross. Early failures to meet high expectations on FMs and FM transfer might have led to what can be called an "FM Winter". However, we think crossing a few other barriers first might make it easier to cope with scepticism in the assurance community and initiate an "FM Spring", at least in assurance practice. The recent successes with FMs for certification of commercial medical devices [75, 126], and the associated burgeoning field of FMs for HCI [173], are examples that could help to overcome this scepticism. Moreover, their is potential for transfer of these results to RAS engineering, where the need for safety assured HCI is also paramount [112, 114].

Threat 3 *The main threat is the reluctance of FM researchers to regularly engage in transfer efforts combined with an exaggerated scepticism of practitioners and with other mechanisms (e.g. lack of funding, poorly focused research evaluation, intellectual property rights) preventing both sides from engaging in an effective bidirectional transfer.*

Remedy 3 *To reduce this threat, building awareness among researchers as well as stronger incentives (e.g. regulation) to provide continuous transfer funding for FM research (e.g. not relying on short-term projects like PhD theses) is needed. A good start on the academic side could then be a specific standardised repository of FM case studies.*

5.4 Too Many Errors in Proofs and Failure to Scale

From the perspective of measurement, Jones and Bonsignour [100, Sec. 4.1] stated that "proofs of correctness sound useful, but [i] errors in the proofs themselves seem to be common failings not covered by the literature. Further, large applications may have thousands of provable algorithms, and [ii] the time required to prove them all might take many years". For [i], the authors opposed 7% of erroneous bug repairs to up to 100% of erroneous proofs, though stating that the latter was based on an anecdote and there had been little data around. Jones and Bonsignour elaborated an example for [ii]: Assuming one provable algorithm per 5 function points[24] and on average 4 proofs per day, Microsoft Windows 7 (160,000 function points) would have about 32,000 provable algorithms, taking a qualified software engineer about 36 calendar years. They highlighted that typically only around 5% of the personnel are trained to do this work, assuming that algorithms and requirements are stable during proof time.

Jones and Bonsignour's argument is based on the view of practical program verification as a largely manual activity. However, research in reasoning about programs has devised promising approaches to proof automation (e.g. bounded model checkers, interactive proof assistants). Assessing such approaches, MacKenzie [125] compared rigorous but manual formal proof with mechanised or automated proof. He could not find a case where mechanised proof threw doubt upon an established mathematical theorem. MacKenzie observed that this underpins the robustness argument of De Millo et al. [35] for proofs as manual social processes rather than lengthy mechanical derivations (cf. Section 3.3). However, opposing De Millo et al.'s view, Jones [101, p. 38] concluded that structuring of knowledge about computing ideas is the actual issue, and not the avoidance of mechanisation. Furthermore, Jones [101, p. 39] mitigated Fetzer's issue of the reality gap in program verification [48] by referring to researchers' continuous efforts in verifying hardware, programming languages, and compilers in addition to individual programs. Finally, in support of Jones and Bonsignour's concerns, Jones [101, p. 38] highlighted the validity of specifications, manual and automated proofs rely on, as a serious issue indeed having received too little attention from researchers (cf. Section 4.4).

Threat 4 *The main threat is the lack of qualified personnel to cope with the required amount and type of proof.*

Remedy 4 *We believe, one angle of attacking this threat is the use of proof assistants* well-integrated *with common environments for requirements specification and software development combined with continuous research and education in the corresponding methods and tools.*

5.5 Failure to Derive Useful Tools

Being loosely related to erroneous proofs, the *information overload through false-positive findings of errors* is a well-known problem in static program analysis. Semi-formal pattern checkers,[25] such as PMD and FindBugs, are exposed to this threat [58]. Additionally, FM-based verification tools, such as Terminator and ESC/Java [49], can be unable to correctly report all potential problems, because they are bounded and therefore unsound. While such tools can be helpful, confronting developers with many irrelevant findings (i.e., false positives) or a high risk of critical misses (i.e., false negatives) can lead to decreased use of FM-based tools.

Figure 2 relates the two information retrieval metrics, *precision* and *recall*, with two adequacy criteria of proof calculi, *soundness* and *completeness*. Precision denotes the ratio of actual (solid circle) to correct (true positive findings in the hashed area) findings, while recall measures the ratio of correct to all theoretically correct (dotted circle) findings. The ideal value for both metrics is 1, meaning that there are only true negatives and true positives. Completeness, although unachievable for richer theories, would correspond to recall and soundness would correspond to a precision of 1.

On the one hand, the usefulness of the calculi underlying FMs is directly proportional only to their completeness and (traditionally) expires with a precision of less than 1. In other words, we usually try to avoid calculi that allow the derivation of false theorems. On the other hand, semi-formal pattern checkers have shown to have a wide, sometimes unacceptable, range of precision and recall of their findings. The usefulness of practical FM-based tools might lie somewhere in the middle between classical calculi and bug finding tools with poor precision/recall values.

Threat 5 *The main threat is the struggle of academia and tool vendors to provide adequate tools, suited to adapt to novel scientific insights, to be integrated with other tools, and to be maintained in a flexible and independent manner.*

Remedy 5 *To reduce this threat, an improvement of education and strong incentives can play an important role here (cf. Sections 5.2 and 5.3).*

[24]A function point is a measure of the conceptual complexity of an IT system relevant for the estimation of the amount of work required to engineer this system.

[25]Also called bug finding tools.

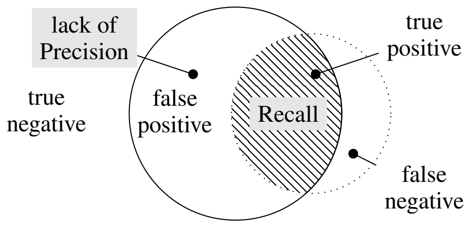

Figure 2: Precision and recall versus soundness and completeness in program verification

6 A Vision of Integrated Formal Methods for Assurance

The following discussion applies to many domains of dependability assurance. However, the complexity of robots and autonomous systems forms a key opportunity for the progress of iFM research and for its successful transfer. Accordingly, Table 1 summarises the discussion in Sections 3 to 5 with an interpretation into RAS assurance practice. Based on the strengths and opportunities described in the Sections 3 and 4, we formulate our vision in terms of working hypotheses:

(H1) From Section 4.1: Software tools for the construction of arguments and production of evidence using *iFMs can meet the challenge* of assuring RAS safe. Computer-assisted assurance cases supported by heterogeneous formal models will increase confidence in their sufficiency, and also aid in maintenance and evolution through modularisation of arguments and evidence.

(H2) From Sections 4.1 and 4.3: iFMs, in particular modern verification tools, will enable *automation of the evidence gathering process*, and highlight potential problems when an assurance case changes or when an incident occurs.

(H3) From Sections 4.1 and 4.3: There is *no stable path to assured autonomy without the use of iFMs*. Acceptable safety will be much more likely achieved with iFMs than without them.

(H4) From Section 3.5: The success of iFMs depends on the ability to *integrate a variety of FMs* for different aspects of RAS (e.g. HCI, safety-security interaction, missing human fallback, environment/world modelling, uncertain prediction/behaviour), which is not currently possible.

(H5) From Sections 3.4 and 3.5: Sophisticated techniques for *model integration and synchronisation are necessary* to support MDE with iFMs. This way, iFMs will make it easier to express consistent RAS models covering all relevant aspects, make their modelling assumptions explicit, and improve future assurance practices.

(H6) From Sections 3.1 to 3.3 and 5.3: iFMs can be beneficial in the short term. However, an important engineering principle is to be conservative [150] and, therefore, not to change procedures unless there is compelling evidence that iFMs are effective. Such evidence can be delivered through *empirical research* [e.g. 98, 151, 163, 177] and collaboration of academia and industry. That evidence *is required to re-evaluate research and foster research progress and transfer*.

(H7) From Section 4.2: The demonstration of cost effectiveness in addition to technical effectiveness of new iFMs is necessary to justify further research.

(H8) From Section 4.4: Norms are a lever of public interest in dependability [47]. Current norms seem to deviate from the state of the art and may fail to guarantee product certification procedures that satisfy the public interest.

Figure 3 assigns these hypotheses to the relationships between foundational and transfer-directed iFM research by example of the RAS domain. Overall, we believe that iFMs have great potential and can improve assurance but practitioners do not use them accordingly.

Opportunity 5 *We could take and enhance credible measures to convince assurance practitioners of our results and effectively transfer these results. For this to happen, we have to answer further research questions.*

7 Empirical, Applied, and Foundational Research

Based on the aforementioned working hypotheses, we state several objectives for foundational and transfer-directed iFM research, formulate research questions, and show our expectations on desirable outcomes of such research.

Table 1: Overview of our SWOT analysis [152] of "iFMs in practical RAS assurance"

Method Strengths: iFMs raise the potential of ... • improvement of RAS models, specification of RAS requirements, automation of RAS verification (Section 3.1) • early detection of systematic errors in RAS designs (Section 3.1) • helpful abstractions to inform engineers about critical RAS properties (Section 3.4) • integration and coordination of several FMs to consistently reason about interrelated RAS properties (Section 3.5) **Community Strengths: iFM research can rely on ...** • many transfer re-entry points from former FM case studies in industrial and academic labs (Section 3.2) • many assurance practitioners who perceive FM usefulness as positive (Section 3.1) • research designs for comparative studies of FM effectiveness (Section 3.3)	**Method Weaknesses: FMs have suffered from ...** • being difficult to learn and apply, many assurance practitioners perceive ease of use of FMs as negative (Section 3.1) • low effectiveness of formal models because of the reality gap (Section 3.4) • fragile effectiveness and productivity in RAS engineering (Section 3.3) **Community Weaknesses: iFM progress has been hampered by ...** • no agreed framework for integration of FMs (Section 3.5) • lack of convincing evidence of FM effectiveness in RAS engineering (Section 3.3) • research ineffectively communicated in iFM teaching/training (Section 3.1)
Key Opportunities for iFM transfer and progress: • The desire for early removal of erroneous RAS behaviour and model-based assurance (Section 4.1) • The desire to learn from RAS accidents and their root causes (Section 4.2) • The desire of RAS assurance to be a mature discipline (Section 4.3) • The desire for adequate and dependable RAS norms (Section 4.4) The analysis provided in this table is an enhancement of the general analysis in Table 3 in Appendix B.	**Method Threats: iFM research is threatened by ...** • misconceptions of semantic unification in RAS assurance (Section 5.1) • iFMs not scaling up to industry-size RASs (Section 5.4) • faulty, tedious, or vacuous proofs (Sections 5.4 and 5.5) • poor integration with RAS engineering tools, processes, and education (Sections 5.2, 5.4, and 5.5) **Transfer Threats: iFM transfer is threatened by ...** • a lack of roboticists' education in iFMs (Sections 5.2 and 5.4) • a lack of iFM researcher engagement in transfer to RAS practice (Section 5.3) • a lack of comprehensive access to quality data from RAS practice (Section 5.3)

7.1 Research Objectives and Tasks

To validate and transfer our research results, we need to

- evaluate how assurance case construction and management for certifiable RASs can be improved by iFMs [57].
- debunk or justify arguments against the use of FMs or FM-based tools in RAS assurance.
- foster a culture of successful FM research transfer to industries performing RAS assurance.

Taking an iFM foundational point of view, we need

- foundational research on the integration and unification of FMs to tackle the complexity of RASs [45].
- a unified semantic foundation for the plethora of notations in RAS assurance, to enable method and tool integration. There are a number of promising research directions still being investigated [89, 158].

Taking an evidence-based point of view, as already highlighted in 1993 by Bowen and Stavridou [18], we need to

- understand the difference between the state of assurance practice and assurance research.

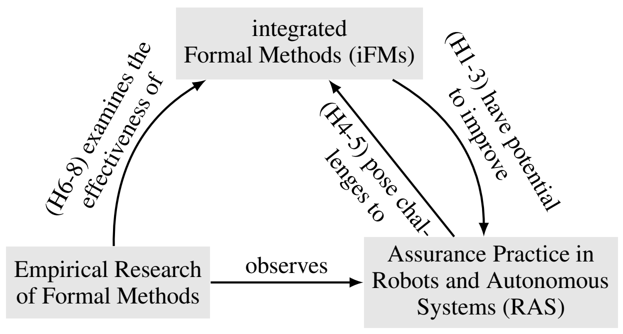
integrated Formal Methods (iFMs)
(H6-8) examines the effectiveness of
(H1-3) have potential to improve
(H4-5) pose challenges to
Empirical Research of Formal Methods
observes
Assurance Practice in Robots and Autonomous Systems (RAS)

Figure 3: Progress of research on integrated formal methods through transfer into and improvement of assurance practice of robots and autonomous systems

- understand in which ways current RAS assurance practices fail, and suggest effective approaches from assurance research. In this way, we can be sure that assurance practice is equipped with state-of-the-art assurance technology for defence against potential liability claims, and that assurance practitioners do not fail in fulfilling their obligations.
- understand how results from assurance research can be validated to be sure that research follows promising directions with high potential of success in assurance practice.

Based on that, we need to

- set concrete directions for empirical FM research in RAS assurance.
- train FM researchers in applying empirical research designs in their work on rigorous assurance cases. Woodcock et al. [177] corroborated this objective by saying that "formal methods champions need to be aware of the need to measure costs".
- avoid biases as found in various branches of scientific research. For example, in the social and biomedical sciences, researchers identified such biases through meta-analyses and suggested measures for bias avoidance [e.g. 44, 50].
- increase the level of evidence of FM research to level 2 according to the hierarchy in [65, Tab. 2].
- avoid knowledge gaps about whether (a) RAS practice is keeping up with the state of the assurance art, and (b) whether recent academic or industrial research is going in the right direction. In this way, we can be sure that we do our best to inform and serve the society.

Using appropriate research designs, we need to

- invite the RAS industry to enhance their efforts in engaging with recent iFM research.
- foster goal-oriented interaction (a) between assurance practitioners and researchers and (b) between FM researchers and assurance researchers. In this way, we can be sure to do everything to keep researchers up to date with respect to practical demands.
- join the FM research and applied assurance research communities (Figure 1 on page 3), both vital for the progress and transfer of assurance research into RAS assurance practice. This way, we can be sure to foster necessary knowledge transfer between these two communities.
- summarise achievements in practical applications of iFMs for constructing assurance cases.
- suggest improvements of curricula for RAS assurance.
- guide the RAS industry in process improvement, training, and tool support.
- guide vendors of FM-based assurance tools to assess and improve their tools and services.

7.2 Some Research Questions addressing these Objectives

The research questions below are relevant for FMs in general. We consider these questions as crucial to be answered for RAS assurance to address the aforementioned objectives:

(Q1) What is the true extent of computer-related accidents up to 2019 [124]? What would these figures mean for the RAS domain?

(Q2) Does the use of formalism detect severe errors to a larger extent than without the use of formalism [151, 163]?

(Q3) Does the use of formalism detect severe errors earlier than without using formalism?

(Q4) Why would such errors be a compelling argument for the use of FMs?

(Q5) Apart from error avoidance and removal, which other benefits of iFMs in practice are evident and can be utilised for method trade-offs?

(Q6) Which criteria play a central role in measuring iFM effectiveness? Particularly, how can FM-based tools be used at scale?

(Q7) How would Commercial-off-the-Shelf and System-Element-out-of-Context verification by iFMs pay off?

(Q8) Which hurdles need to be overcome to use iFMs in practice to the maximum benefit?

(Q9) How do we know when these hurdles are actually overcome?

(Q10) How can FMs (from different disciplines) be used together (iFMs, unification)?

(Q11) How can FMs be used to assure systems (e.g. computer vision in road vehicles) involving *artificial intelligence* (AI) techniques like machine learning and deep neural networks?

(Q12) How can FMs be integrated into assurance cases to support certification against international safety and security standards?

(Q13) How can we best combine formal and informal methods? For example, how can we deal with the issue of the validity of specifications that proofs rely on [101].

(Q14) How can we best present formal requirements, evidence, and artefacts in an assurance case?

(Q15) How can empirical research help in successfully demonstrating the capabilities of iFMs for rigorous and certifiable autonomy assurance?

This list of research questions can easily be extended by further more detailed empirical questions from the settings discussed in [100, Sec. 4.4].

7.3 Some Envisaged Research Outcomes

Our vision of *rigorous RAS assurance* implies foundational iFM research to result in

- novel semantic frameworks unifying best practice methods, models, and formalisms established in RAS
- new concepts for iFM-based development environments
- new computational theories to support formal modelling and verification of RAS
- evaluations of
  - assurance tools, languages, frameworks, or platforms used in practice regarding their support of iFMs
  - the integration of iFMs into modelling and programming techniques, assurance methods, and assurance processes
  - languages for linking informal requirements with evidence from iFMs
  - (automated) abstraction techniques used in assurance and certification
- opinions, positions, and visions on FM integration and unification for rigorous practical assurance.

Our vision of *rigorous RAS assurance* implies applied and empirical iFM research to result in

- comparisons of
  - projects applying iFMs in assurance practice with similar practical projects applying non-iFM approaches
  - iFM-based (embedded software) assurance with assurance approaches in traditional engineering disciplines
- checklists, metrics, and benchmarks (for and beyond tool performance) for
  - the evaluation and comparison of iFM-based assurance approaches (e.g. confidence level)

  - relating error removal and incident root cause data (e.g. efficiency and effectiveness in removal of severe errors or in avoidance of severe accidents, cf. [124])
  - usability and maturity assessment of iFMs (e.g. abstraction effort, proof complexity, assurance case complexity, productivity)
  - the evaluation of FM budget cases (cf. [33] in electronic hardware development).
- experiences in or surveys (e.g. systematic mappings and reviews of assurance case research, interview studies with assurance practitioners, cf. Appendix A) of
  - iFM transfers and applications (e.g. case studies in assurance and certification projects)
  - challenges, limitations/barriers, and benefits of iFMs in assurance and certification projects,
- research designs (e.g. for controlled field experiments) for the practical validation of iFMs in assurance and certification projects
- opinions, positions, and visions on future research, education, and training in the use of iFMs in assurance and certification.

8 Summary

Along the lines of Hoare et al. [90], we analysed strengths, weaknesses, opportunities, and threats to determine the potential of integrated formal methods to improve the practice of dependability assurance. Emphasising robots and autonomous systems as an area in the spotlight of dependability assurance, we express our expectations of research progress and transfer. From these expectations, we derived a research and research transfer agenda with the objective of (i) enhancing the foundations of integrated formal methods, (ii) collecting evidence on the effectiveness of integrated formal methods in practice, (iii) successfully transferring integrated formal methods into the assurance practice, with a short-term focus on robots and autonomous systems, and (iv) fostering research progress, education, and training from the results of this transfer effort.

Acknowledgements This work is partly supported by the Deutsche Forschungsgemeinschaft (DFG) under the Grant no. 381212925 and the EPSRC projects CyPhyAssure,[26] grant reference EP/S001190/1, and RoboCalc, grant reference EP/M025756/1. We are deeply grateful to Michael Butler, Ana Cavalcanti, John Fitzgerald, Cliff Jones, and Peter Gorm Larsen for helpful feedback on previous versions of this manuscript and for their genuine support in shaping the direction of this work.

References

[1] Abrial, J.-R. (1996). *The B-book: assigning programs to meanings*. Cambridge University Press.

[2] Aichernig, B. K. and T. Maibaum (Eds.) (2003). *Formal Methods at the Crossroads. From Panacea to Foundational Support*. Springer Berlin Heidelberg.

[3] Akdur, D., V. Garousi, and O. Demirörs (2018). A survey on modeling and model-driven engineering practices in the embedded software industry. *Journal of Systems Architecture 91*, 62–82.

[4] Araki, K., A. Galloway, and K. Taguchi (Eds.) (1999). *Proc. 1st Intl. Conf. on Integrated Formal Methods*. Springer.

[5] Aravantinos, V., S. Voss, S. Teufl, F. Hölzl, and B. Schätz (2015). Autofocus 3: Tooling concepts for seamless, model-based development of embedded systems. In *ACES-MB&WUCOR@MoDELS*, Volume 1508 of *CEUR Workshop Proceedings*, pp. 19–26. CEUR-WS.org.

[6] Austin, S. and G. Parkin (1993, March). Formal methods: A survey. Technical report, National Physical Laboratory, Teddington, Middlesex, UK.

[7] Avizienis, A., J.-C. Laprie, B. Randell, and C. Landwehr (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on 1*(1), 11–33.

[8] Back, R. J. R. and J. von Wright (1989). Refinement calculus, part i: Sequential nondeterministic programs. In *Proc. REX Workshop*, Volume 430 of *LNCS*. Springer.

[9] Barroca, L. M. and J. A. McDermid (1992). Formal methods: Use and relevance for the development of safety-critical systems. *Comp. J. 35*(6), 579–99.

[10] Ben-David, S., P. Hrubeš, S. Moran, A. Shpilka, and A. Yehudayoff (2019). Learnability can be undecidable. *Nature Machine Intelligence 1*(1), 44–48.

[26] CyPhyAssure Project: https://www.cs.york.ac.uk/circus/CyPhyAssure/.

[11] Berry, D. M. and W. F. Tichy (2003). Comments on "Formal methods application: an empirical tale of software development". *IEEE Transactions on Software Engineering 29*(6), 567–571.

[12] Bicarregui, J. C., J. S. Fitzgerald, P. G. Larsen, and J. Woodcock (2009). Industrial practice in formal methods: A review. In A. Cavalcanti and D. R. Dams (Eds.), *FM 2009: Formal Methods*, Berlin, Heidelberg, pp. 810–813. Springer Berlin Heidelberg.

[13] Bonini, C. P. (1962). *Simulation of information and decision systems in the firm*. Prentice-Hall.

[14] Börger, E., M. Butler, J. P. Bowen, and P. Boca (Eds.) (2008). *Abstract State Machines, B and Z*, Volume 5238 of *LNCS*. Springer Berlin Heidelberg.

[15] Boulanger, J.-L. (2012). *Industrial Use of Formal Methods: Formal Verification*. Wiley-ISTE.

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

[17] Bowen, J. and S. Reeves (2017, 6). Generating obligations, assertions and tests from UI models. In *Proc. ACM Symposium on Engineering Interactive Computing Systems (EICS)*, Volume 1. ACM.

[18] Bowen, J. and V. Stavridou (1993). Safety-critical systems, formal methods and standards. *Software Engineering Journal 8*(4), 189.

[19] Box, G. E. P. and N. R. Draper (1986). *Empirical model-building and response surfaces*. Wiley.

[20] Breu, R., U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe, and V. Thurner (1997). Towards a formalization of the unified modeling language. In *ECOOP'97 – Object-Oriented Programming*, pp. 344–366. Springer Berlin Heidelberg.

[21] Brooks, R. A. (1992). Artificial life and real robots. In F. J. Varela and P. Bourgine (Eds.), *ECAL*, pp. 3–10. MIT Press.

[22] Broy, M. (2006). Challenges in automotive software engineering. In *Proceedings of the 28th International Conference on Software Engineering - ICSE'06*. ACM Press.

[23] Broy, M. (2017). A logical approach to systems engineering artifacts: semantic relationships and dependencies beyond traceability—from requirements to functional and architectural views. *Software & Systems Modeling 17*(2), 365–393.

[24] Broy, M. (2018). Yesterday, today, and tomorrow: 50 Years of software engineering. *IEEE Software 35*(5), 38–43.

[25] Broy, M., M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu (2010). Seamless model-based development: From isolated tools to integrated model engineering environments. *Proceedings of the IEEE 98*(4), 1–21.

[26] Broy, M. and O. Slotosch (1998). Enriching the software development process by formal methods. In *Applied Formal Methods – FM-Trends 98*, Volume 1641 of *LNCS*, pp. 44–61. Springer.

[27] Butterfield, A. and P. Gancarski (2009, 11). The denotational semantics of slotted-Circus. In A. Cavalcanti and M. Damm (Eds.), *Formal Methods 2009*, Volume 5850 of *LNCS*, Eindhoven, Netherlands. Springer.

[28] Cataño, N. and M. Huisman (2002). Formal specification and static checking of gemplus' electronic purse using ESC/java. In *FME 2002:Formal Methods—Getting IT Right*, pp. 272–289. Springer Berlin Heidelberg.

[29] Chong, S., J. Guttman, A. Datta, A. Myers, B. Pierce, P. Schaumont, T. Sherwood, and N. Zeldovich (2016). Report on the NSF workshop on formal methods for security. Technical report, National Science Foundation.

[30] Chudnov, A., N. Collins, B. Cook, J. Dodds, B. Huffman, C. MacCárthaigh, S. Magill, E. Mertens, E. Mullen, S. Tasiran, A. Tomb, and E. Westbrook (2018). Continuous formal verification of Amazon s2n. In *Computer Aided Verification*, pp. 430–446. Springer International Publishing.

[31] Clarke, E. M. and J. M. Wing (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys 28*(4), 626–643.

[32] Cruanes, S., G. Hamon, S. Owre, and N. Shankar (2013). Tool integration with the evidential tool bus. In *Verification, Model Checking and Abstract Interpretation (VMCAI)*, Volume 7737 of *LNCS*. Springer.

[33] Darbari, A. (2018). The budget case for formal verification.

[34] Daylight, E. G. (2013, 2). From mathematical logic to programming-language semantics: a discussion with tony hoare. *Journal of Logic and Computation 25*(4), 1091–1110.

[35] De Millo, R. A., R. J. Lipton, and A. J. Perlis (1979, 5). Social processes and proofs of theorems and programs. *Communications of the ACM 22*(5), 271–280.

[36] Debruyne, V., F. Simonot-Lion, and Y. Trinquet (2004). EAST-ADL — an architecture description language. In *IFIP The International Federation for Information Processing*, pp. 181–195. Springer-Verlag.

[37] Denney, E. and G. Pai (2018). Tool support for assurance case development. *Automated Software Engineering 25*, 435–499.

[38] Dijkstra, E. W. (1978, 4). On a political pamphlet from the middle ages. *ACM SIGSOFT Software Engineering Notes 3*(2), 14–16.

[39] Dunne, S. and B. Stoddart (Eds.) (2006). *1st Intl. Symp. on Unifying Theories of Programming*, Volume 4010 of *LNCS*. Springer.

[40] Endres, A. (1975). An analysis of errors and their causes in system programs. *IEEE Transactions on Software Engineering SE-1*(2), 140–149.

[41] Erdogmus, H., N. Medvidovic, and F. Paulisch (2018). 50 Years of software engineering. *IEEE Software 35*(5), 20–24.

[42] Evans, A. S. (1994). Specifying and verifying concurrent systems using z. In *Proc. 2nd. Intl. Symp on Formal Methods Europe*, Volume 873 of *LNCS*. Springer.

[43] Evans Data (2018). Global developer population and demographic study. Technical Report Volume 1, Evans Data Corporation.

[44] Fanelli, D., R. Costas, and J. P. A. Ioannidis (2017, 3). Meta-assessment of bias in science. *Proceedings of the National Academy of Sciences 114*(14), 3714–3719.

[45] Farrell, M., M. Luckcuck, and M. Fisher (2018). Robotics and integrated formal methods: Necessity meets opportunity. In *Proc. 14th. Intl. Conf. on Integrated Formal Methods (iFM)*, Volume LNCS 11023, pp. 161–171. Springer.

[46] Feiler, P. H., B. Lewis, S. Vestal, and E. Colbert (2004). An overview of the SAE architecture analysis & design language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering. In *IFIP The International Federation for Information Processing*, pp. 3–15. Springer-Verlag.

[47] Feitelson, D. G. (2019). Tony's law. *Communications of the ACM 62*(2), 28–31.

[48] Fetzer, J. H. (1988, 8). Program verification: the very idea. *Communications of the ACM 31*(9), 1048–1063.

[49] Flanagan, C., K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe, and R. Stata (2002). Extended static checking for java. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation - PLDI'02*. ACM Press.

[50] Franco, A., N. Malhotra, and G. Simonovits (2014). Publication bias in the social sciences: Unlocking the file drawer. *Science 345*, 1502–4.

[51] Galloway, A. and B. Stoddart (1997a, 11). An operational semantics for zccs. In *Proc. 1st Intl. Conf. on Formal Engineering Methods*. IEEE Computer Society.

[52] Galloway, A. J., T. J. Cockram, and J. A. McDermid (1998). Experiences with the application of discrete formal methods to the development of engine control software. *IFAC Proceedings Volumes 31*(32), 49–56.

[53] Galloway, A. J. and B. Stoddart (1997b). Integrated formal methods. In *Proc. INFORSID*. INFORSID.

[54] Giese, M. and R. Heldal (2004). From informal to formal specifications in UML. *The Unified Modelling Language*, 197–211.

[55] Glass, R. L. (2002). *Facts and Fallacies of Software Engineering*. Pearson Education (US).

[56] Gleirscher, M. (2014). *Behavioral Safety of Technical Systems*. Dissertation, Technische Universität München.

[57] Gleirscher, M., S. Foster, and Y. Nemouchi (2019). Evolution of formal model-based assurance cases for autonomous robots. In *17th Int. Conf. Software Engineering and Formal Methods*.

[58] Gleirscher, M., D. Golubitskiy, M. Irlbeck, and S. Wagner (2014). Introduction of static quality analysis in small and medium-sized software enterprises: Experiences from technology transfer. *Software Quality Journal 22*(3), 499–542.

[59] Gleirscher, M. and D. Marmsoler (2018). Formal methods in dependable systems engineering: A survey of professionals from Europe and North America. Working paper, Department of Computer Science, University of York.

[60] Gleirscher, M. and A. Nyokabi (2018). System safety practice: An interrogation of practitioners about their activities, challenges, and views with a focus on the European region. Working paper, Department of Computer Science, University of York.

[61] Gleirscher, M., D. Ratiu, and B. Schätz (2007). Incremental integration of heterogeneous systems views. In *1st Int. Conf. Systems Engineering and Modeling, ICSEM 2007, Herzliyya-Haifa, Israel, March 20-23, 2007*, pp. 50–9.

[62] Gnesi, S. and T. Margaria (2013). *Formal Methods for Industrial Critical Systems: A Survey of Applications*. Wiley-IEEE Press.

[63] Godefroid, P., N. Klarlund, and K. Sen (2005). Dart: Directed automated random testing. In *Programming Language Design and Implementation (PLDI)*, Volume 40 of *ACM SIGPLAN Notices*, pp. 213–223. ACM.

[64] Godefroid, P., M. Y. Levin, and D. Molnar (2012). SAGE: Whitebox fuzzing for security testing. *Communications of the ACM 55*(3).

[65] Goues, C. L., C. Jaspan, I. Ozkaya, M. Shaw, and K. T. Stolee (2018). Bridging the gap: From research to practical advice. *IEEE Software 35*(5), 50–57.

[66] Gray, J. and E. Brewer (2001). Dependability in the internet era. In *Proceedings of the High Dependability Computing Consortium Conference*.

[67] Graydon, P. J. (2015, 6). Formal assurance arguments: A solution in search of a problem? In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 517–528.

[68] Greenwell, W. S., J. C. Knight, C. M. Holloway, and J. J. Pease (2006). A taxonomy of fallacies in system safety arguments. In *24th International System Safety Conference*.

[69] Grieskamp, W., T. Santen, and B. Stoddart (Eds.) (2000). *Proc. 2nd Intl. Conf. on Integrated Formal Methods*, Volume 1945 of *LNCS*. Springer Berlin Heidelberg.

[70] Guiochet, J., M. Machin, and H. Waeselynck (2017). Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems 94*.

[71] Gunter, C. A., E. L. Gunter, M. Jackson, and P. Zave (2000). A reference model for requirements and specifications. *IEEE Software 17*(3), 37–43.

[72] Habli, I., I. Ibarra, R. Rivett, and T. Kelly (2010, 4). Model-based assurance for justifying automotive functional safety. In *Proc. SAE World Congress*.

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

[74] Harel, D. and B. Rumpe (2004). Meaningful modeling: What's the semantics of "Semantics"? *IEEE Computer 37*(10), 64–72.

[75] Harrison, M. D., L. Freitas, M. Drinnan, J. C. Campos, P. Masci, C. Maria, and M. Whitaker (2019, 2). Formal techniques in the safety analysis of software components of a new dialysis machine. *Science of Computer Programming 175*, 17–34.

[76] Harrison, M. D., P. Masci, and J. C. Campos (2018). Formal modelling as a component of user centred design. In *Proc. Software Technologies: Applications and Foundations (STAF)*, Volume 11176 of *LNCS*, pp. 274–289. Springer.

[77] Hawkins, R., I. Habli, D. Kolovos, R. Paige, and T. Kelly (2015). Weaving an assurance case from design: A model-based approach. In *Proc. 16th IEEE Intl. Conf. on High Assurance Systems Engineering (HASE)*. IEEE.

[78] Health and Safety Executive (2003). *Out of Control*. HSE Books.

[79] Hehner, E. C. R. (1984). Predicative programming. *Communications of the ACM 27*(2), 134–151.

[80] Hehner, E. C. R. (1990). A practical theory of programming. *Science of Computer Programming 14*, 133–158.

[81] Hehner, E. C. R. and C. A. R. Hoare (1983). A more complete model of communicating processes. *Theoretical Computer Science 26*, 105–120.

[82] Hehner, E. C. R. and A. J. Malton (1988). Termination conventions and comparative semantics. *Acta Informatica 25*.

[83] Heitmeyer, C., A. Bull, C. Gasarch, and B. Labaw (1995). SCR: a toolset for specifying and analyzing requirements. In *Proceedings of the Tenth Annual Conference on Computer Assurance, Systems Integrity, Software Safety, and Process Security - COMPASS'95*. IEEE.

[84] Hinchey, M., M. Jackson, P. Cousot, B. Cook, J. P. Bowen, and T. Margaria (2008). Software engineering and formal methods. *Communications of the ACM 51*(9), 54–59.

[85] Hoare, C. A. R. (1981). The Emperor's Old Clothes. *Communications of the ACM 24*(2), 75–83. The 1980 ACM Turing Award Lecture.

[86] Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

[87] Hoare, C. A. R. (1994, 7). Unified theories of programming. Technical report, Oxford Computing Laboratory. Also published by Springer in *Mathematical Methods in Program Development*, 1997.

[88] Hoare, C. A. R. (1996). How did software get so reliable without proof? In *FME'96: Industrial Benefit and Advances in Formal Methods*, pp. 1–17. Springer Berlin Heidelberg.

[89] Hoare, C. A. R. and J. He (1998). *Unifying Theories of Programming*. Pearson College Div.

[90] Hoare, C. A. R., J. Misra, G. T. Leavens, and N. Shankar (2009). The verified software initiative. *ACM Computing Surveys 41*(4), 1–8.

[91] Hoffmann, L. (2018). Promoting common sense, reality, dependable engineering. *Communications of the ACM 61*(12), 128–129.

[92] Höhn, R. and S. Höppner (2008). *Das V-Modell XT: Grundlagen, Methodik und Anwendungen*. Springer.

[93] Holloway, C. M. and C. W. Johnson (2008, 10). How past loss of control accidents may inform safety cases for advanced control systems on commercial aircraft. In *3rd IET International Conference on System Safety*, pp. 1–6.

[94] Hopkins, A. (2004). Quantitative risk assessment: A critique. Working Paper 25, Australian National University.

[95] Huber, F., B. Schätz, A. Schmidt, and K. Spies (1996). AutoFocus — a tool for distributed systems specification. In *Lecture Notes in Computer Science*, pp. 467–470. Springer Berlin Heidelberg.

[96] Jackson, D., L. I. Millett, and M. Thomas (2007). *Software for dependable systems: sufficient evidence?* National Academies Press.

[97] Jakobi, N., P. Husbands, and I. Harvey (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *ECAL*, Volume 929 of *LNCS*. Springer.

[98] Jeffery, R., M. Staples, J. Andronick, G. Klein, and T. Murray (2015). An empirical research agenda for understanding formal methods productivity. *Information and Software Technology 60*, 102–112.

[99] Jhala, R., R. Majumdar, R. Alur, A. Datta, D. Jackson, S. Krishnamurthi, J. Regehr, N. Shankar, and C. Tinelli (2012). Formal methods: Future directions & transition to practice. Workshop report, National Science Foundation.

[100] Jones, C. and O. Bonsignour (2011). *The Economics of Software Quality*. Addison-Wesley Professional.

[101] Jones, C. B. (2003, 4). The early search for tractable ways of reasoning about programs. *IEEE Annals of the History of Computing 25*(2), 26–49.

[102] Kalra, N. and S. M. Paddock (2016). Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? Research Report RR-1478-RC, RAND Corp.

[103] Kästner, D., S. Wilhelm, S. Nenova, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival (2010). Astrée: Proving the absence of runtime errors. In *Proc. of Embedded Real Time Software and Systems (ERTS2)*, Volume 9.

[104] Kelly, T. P. (1999). *Arguing Safety – A Systematic Approach to Safety Case Management*. Ph. D. thesis, University of York, Dept. of Computer Science.

[105] Kim, M., Y. Kim, and Y. Choi (2011). Concolic testing of the multi-sector read operation for flash storage platform software. *Formal Aspects of Computing 24*, 355–374.

[106] Kitchenham, B. and S. L. Pfleeger (1996). Software quality: The elusive target. *IEEE Software 13*(1), 12–21.

[107] Klein, G., J. Andronick, M. Fernandez, I. Kuz, T. Murray, and G. Heiser (2018). Formally verified software in the real world. *Communications of the ACM 61*(10), 68–77.

[108] Knight, J. C. (2002). Safety critical systems: Challenges and directions. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, New York, NY, USA, pp. 547–50. ACM.

[109] Knight, J. C. (2014). Safety standards – A new approach. In *System Safety Symposium*.

[110] Knight, J. C., C. L. DeJong, M. S. Gibble, and L. G. Nakano (1997). Why are formal methods not used more widely? In *Fourth NASA Formal Methods Workshop*, pp. 1–12.

[111] Koopman, P. (2014). A case study of Toyota unintended acceleration and software safety.

[112] Koopman, P. and M. Wagner (2017, 1). Autonomous vehicle safety: An interdisciplinary challenge. *IEEE Intelligent Transportation Systems Magazine 9*(1), 90–96.

[113] Kronlöf, K. (Ed.) (1993). *Method Integration: Concepts and Case Studies*. Wiley.

[114] Kun, A. L., S. Boll, and A. Schmidt (2016, 1). Shifting gears: User interfaces in the age of autonomous driving. *IEEE Pervasive Computing 15*(1), 32–38.

[115] Ladkin, P. B. (2013a). Root Cause Analysis: Terms and definitions, AcciMaps, MES, SOL and WBA. Technical report, University of Bielefeld.

[116] Ladkin, P. B. (2013b). Standards for standards improving the process.

[117] Lee, E. A. and M. Sirjani (2018). What good are models? In *Formal Aspects of Component Software (FACS)*, Volume 11222 of *LNCS*. Springer.

[118] Leino, K. R. M. (2017). Accessible software verification with dafny. *IEEE Software 34*(6), 94–97.

[119] Liebel, G., N. Marko, M. Tichy, A. Leitner, and J. Hansson (2016). Model-based engineering in the embedded systems domain: An industrial survey on the state-of-practice. *Software & Systems Modeling 17*(1), 91–113.

[120] Littlewood, B., I. Bainbridge, and R. E. Bloomfield (1998). The use of computers in safety-critical applications.

[121] Lozano-Perez, T. and M. A. Wesley (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM 22*(10), 560–570.

[122] Luckcuck, M., M. Farrell, L. Dennis, C. Dixon, and M. Fisher (2018). Formal specification and verification of autonomous robotic systems: A survey. *ArXiv e-prints*.

[123] Lutz, R. R. (1993). Analyzing software requirements errors in safety-critical, embedded systems. In *IEEE Int. Symp. Req. Eng.*, pp. 126–33. IEEE.

[124] MacKenzie, D. A. (1994). Computer-related accidental death: an empirical exploration. *Science and Public Policy 21*(4), 233–248.

[125] MacKenzie, D. A. (2001). *Mechanizing Proof: Computing, Risk, and Trust*. Inside Technology. The MIT Press.

[126] Masci, P., P. Curzon, M. D. Harrison, A. Ayoub, I. Lee, and H. Thimbleby (2013, 6). Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example. In *Proc. ACM Symposium on Engineering Interactive Computing Systems (EICS)*, pp. 81–90. ACM.

[127] Mendez-Fernandez, D., B. Penzenstadler, M. Kuhrmann, and M. Broy (2010). A meta model for artefact-orientation: Fundamentals and lessons learned in requirements engineering. In *MODELS'10*.

[128] Meng, M. and A. C. Kak (1993). Mobile robot navigation using neural networks and nonmetrical environment models. *IEEE Control Systems 13*(5).

[129] Miller, S. P. (1998). The industrial use of formal methods: was darwin right? In *Proceedings. 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*. IEEE Comput. Soc.

[130] Miller, S. P., D. A. Greve, M. M. Wilding, and M. Srivas (1999). Formal verification of the aamp-fv microcode. Technical Report NASA/CR-1999-208992, NASA.

[131] Miller, S. P., M. W. Whalen, and D. D. Cofer (2010). Software model checking takes off. *Communications of the ACM 53*(2), 58–64.

[132] Milner, R. (1989). *Communication and Concurrency*. Prentice Hall.

[133] Milner, R. (1999). *Communicating and mobile systems: the π-calculus*. Cambridge University Press.

[134] Mohagheghi, P., W. Gilani, A. Stefanescu, and M. A. Fernandez (2012). An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empirical Software Engineering 18*(1), 89–116.

[135] Moore, A., T. O'Reilly, P. D. Nielsen, and K. Fall (2016). Four thought leaders on where the industry is headed. *IEEE Software 33*(1), 36–39.

[136] Morgan, C. (1996, 1). *Programming from Specifications*. Prentice-Hall.

[137] Morris, J. M. (1987). A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming 9*(3), 287–306.

[138] Motor Industry Research Association (1994). *Development Guidelines for Vehicle Based Software*. Motor Industry Research Association.

[139] Naur, P. (1994, 3). Proof versus formalization. *BIT 34*(1), 148–164.

[140] Neumann, P. G. (1995). *Computer-related Risks*. NY, USA: Addison-Wesley.

[141] Neumann, P. G. (2018). Risks to the public. *ACM SIGSOFT Software Engineering Notes 43*(2), 8–11.

[142] O'Hearn, P. W. (2018). Continuous reasoning. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS'18*. ACM Press.

[143] Oliveira, M., A. Cavalcanti, and J. Woodcock (2009). A UTP semantics for Circus. *Formal Aspects of Computing 21*, 3–32.

[144] Oliveira, R., S. Dupuy-Chessa, and G. Calvary (2015, 6). Plasticity of user interfaces: Formal verification of consistency. In *Proc. ACM Symposium on Engineering Interactive Computing Systems (EICS)*, pp. 260–265. ACM.

[145] Paige, R. F. (1997). A meta-method for formal method integration. In *Proc. 4th. Intl. Symp. on Formal Methods Europe (FME)*, Volume 1313 of *LNCS*, pp. 473–494. Springer.

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

[147] Parnas, D. L. (2017). The real risks of artificial intelligence. *Communications of the ACM 60*(10), 27–31.

[148] Parnas, D. L. and J. Madley (1995). Function documents for computer systems. *Science of Computer Programming 25*, 41–61.

[149] Peleska, J. and W.-L. Huang (2016). Industrial-strength model-based testing of safety-critical systems. In *FM 2016: Formal Methods*, pp. 3–22. Springer International Publishing.

[150] Perlis, A. J. (1969). Identifying and developing curricula in software engineering. In *Proceedings of the May 14-16, 1969, Spring Joint Computer Conference*, AFIPS '69 (Spring), New York, NY, USA, pp. 540–541. ACM.

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

[152] Piercy, N. and W. Giles (1989). Making SWOT analysis work. *Marketing Intelligence & Planning 7*(5/6), 5–7.

[153] Posse, E. and J. Dingel (2016). An executable formal semantics for UML-RT. *Software & Systems Modeling 15*(1), 179–217.

[154] Rae, A. J., M. Nicholson, and R. D. Alexander (2010). The state of practice in system safety research evaluation. In *5th IET International Conference on System Safety 2010*. IET.

[155] Ramesh, B. and M. Jarke (2001). Towards Reference Models for Requirements Traceability. *IEEE Trans. Soft. Eng. 27*(1), 58–93.

[156] Randell, B. (2018). Fifty years of software engineering - or - the view from garmisch. *CoRR abs/1805.02742*. ICSE keynote speech.

[157] Roscoe, A. W., J. Woodcock, and L. Wulf (1994). Non-interference through determinism. In *ESORICS 94*, Volume 875 of *LNCS*. Springer.

[158] Rosu, G. and T. F. Serbanuta (2010). An overview of the K semantic framework. *The Journal of Logic and Algebraic Programming (JLAP) 79*(6), 397–434.

[159] RTCA DO-333 (2012). Formal methods supplement to DO-178C and DO-278A. Standard, Radio Technical Commission for Aeronautics (RTCA).

[160] Rushby, J. (2013). Logic and epistemology in safety cases. In *Lecture Notes in Computer Science*, Volume 8153, pp. 1–7. Springer Berlin Heidelberg.

[161] Rushby, J. (2014). Mechanized support for assurance case argumentation. In *New Frontiers in Artificial Intelligence*, Volume 8417 of *LNCS*. Springer.

[162] Schaffer, K. and J. Voas (2016). What happened to formal methods for security? *Computer 49*(8), 70–79.

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

[164] Sobel, A. E. K. and M. R. Clarkson (2003). Response to comments on "Formal methods application: An empirical tale of software development". *IEEE Transactions on Software Engineering 29*(6), 572–575.

[165] Spivey, J. M. (1989). *The Z-Notation - A Reference Manual*. Englewood Cliffs, N. J.: Prentice Hall.

[166] Stepney, S., D. Cooper, and J. Woodcock (2000, July). An electronic purse: Specification, refinement, and proof. Technical monograph PRG-126, Oxford University Computing Laboratory.

[167] Stevens, P. (2018). Is bidirectionality important? In *European Conference on Modelling Foundations and Applications (ECMFA)*, Volume 10890 of *LNCS*, pp. 1–11. Springer.

[168] van Glabbeek, R. J. (2001). *Handbook of Process Algebra*, Chapter 1. "The Linear Time - Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes", pp. 3–99. Elsevier.

[169] van Lamsweerde, A. (2000). Formal specification: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, New York, NY, USA, pp. 147–159. ACM.

[170] van Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley.

[171] Wei, K., J. Woodcock, and A. Cavalcanti (2013). Circus Time with Reactive Designs. In *Unifying Theories of Programming*, Volume 7681 of *LNCS*, pp. 68–87. Springer.

[172] Wei, R., T. Kelly, X. Dai, S. Zhao, and R. Hawkins (2019, 8). Model based system assurance using the structured assurance case metamodel. *Journal of Software and Systems 154*, 211–233.

[173] Weyers, B., J. Bowen, A. Dix, and P. Palanque (Eds.) (2017). *The Handbook of Formal Methods for Human-Computer Interaction*. HCIS. Springer.

[174] Whitehead, J. (2007, 5). Collaboration in software engineering: A roadmap. In *Future of Software Engineering (FOSE'07)*. IEEE.

[175] Wieringa, R. and E. Dubois (1994). Integrating semi-formal and formal software specification techniques. *Information Systems 19*(4), 33–54.

[176] Woodcock, J. (2006, 10). First steps in the Verified Software grand challenge. *IEEE Computer 39*(10).

[177] Woodcock, J., P. G. Larsen, J. Bicarregui, and J. Fitzgerald (2009). Formal methods: Practice and experience. *ACM Computing Surveys 41*(4), 19:1–19:36.

[178] Woodcock, J. and C. Morgan (1990). Refinement of state-based concurrent systems. In *3rd Intl. Symp. of VDM Europe*, Volume 428 of *LNCS*, pp. 340–351. Springer.

Appendix

A Studying Formal Method Practitioners: A Ballpark Figure

In this section, we make a brief excursion to the relationship between FMs and *formal inspection* (FI), closing with a rough estimate of the size of the population of FM users and the meaning of such an estimate for the empirical study of that population. Such studies include cross-sectional and longitudinal surveys based on questionnaires or semi-structured interviews.

Formal Inspection versus Formal Methods FI encompasses a variety of techniques (e.g. peer reviews, walk-throughs) where critical process artefacts (e.g. program code) are checked (e.g. manually or using software tools) against a variety of criteria (e.g. checklists), usually by a group of independent qualified engineers. For the sake of simplicity of the following discussion, we assume that FMs can be seen as a particularly rigorous variant of FI where formal specifications serve as a particular way of formulating checklists.

We compare the use of FMs with the use of FI. According to Jones and Bonsignour [100, Sec. 4.4], formal inspections are used in more than 35% of commercial defence, systems, and embedded software projects, and FMs are estimated to be applied in less than 1% of overall commercial software engineering projects. To get an idea of this coverage data, we perform an analysis of the global embedded software market based on other global software market indicators in Table 2. We found estimates of systems and software professionals world-wide and estimates of annual US business values.[27]

A uniform distribution would entail roughly 37000 USD/year per person in the general software domain and 10000 USD/year per person in the embedded software domain. Clearly, geographically strongly differing salaries and part-time engagement rule out a uniform distribution, yet providing figures helpful for our purposes.

Next, we apply the following proportions: From a worldwide population of around 18.5 million software developers in 2014 [43], about 19% live in the US, 10% in China, 9.8% in India, 36% Asia/Pacific region, 39 % live in Europe, the Middle East, and Africa; and 30% in the Americas.[28] The *design to quality assurance* (i.e., verification and test) *cost ratio* is observed to be approximately $30 : 70$.[29] About 20% of embedded software personnel are quality assurance engineers (i.e., test, verification, or validation engineers).[30]

The estimates in Table 2 suggest that around 2% of the overall pure software market are allocated to the embedded pure software market. 35% coverage of formal inspection in about 13.5% of the overall software market $(161/(689{+}515) = 0.134)$ would result in roughly 4.7% coverage of all software projects by formal inspection versus at most 1% coverage by FMs. However, from this data we can hardly know whether rates of FM use get close to or beyond 10% in high-criticality systems projects.

Assuming that in about 35% of embedded software projects the quality assurance personnel would use formal inspection and that in every fifth $(1 : 4.7)$ of such projects formal methods would be used, the current population of regular practical FM users would globally amount to about 5040 $(= 72,000 * 0.34 * 0.20)$ persons. Note that these numbers are rough estimates. However, we believe their order of magnitude is realistic. Given that these persons would on average earn about 100,000 USD/year each, we would speak of an annual business value of around USD 504 million.

Empirical Studies of FI and FM Practitioners Importantly, from these data we can determine minimum sample sizes for surveys. For example, assume we want to have 95% confidence in our test results and are fine with a confidence interval of $\pm 7\%$. Then, for *regular practical FM users*, a population of the size of 5040 persons would require us to sample 189 independent data points (e.g. questionnaire responses). The population of *regular practical FI users*, 25200 $(= 72,000 * .35)$ would imply a minimum sample size of 194.

For any sample of such size, any survey has to argue why the sample *represents* the population. This step depends on the possibilities given during the sampling stage. Obviously, reaching out to 189 out of 5040 persons whose locations might be largely unknown is an extremely difficult task that might only be tackled in terms of a global group effort among researchers and practitioners. Given that this challenge can be tackled, these figures determine the type of surveys required for the collection of confident evidence. Moreover, these figures limit the manual effort to perform and repeat valuable qualitative studies to a reasonable amount.

[27]See https://en.wikipedia.org/wiki/Software_industry and https://softwareengineering.stackexchange.com/questions/18959/what-proportion-of-prog

[28]See https://adtmag.com/Blogs/WatersWorks/2014/01/Worldwide-Developer-Count.aspx.

[29]See https://www.slideshare.net/pboulet/socdesign.

[30]See https://de.slideshare.net/vdcresearch/searching-for-the-total-size-of-the-embedded-software-engineering-market.

Table 2: Data for the estimation of the size of the formal inspection and formal method market

Global market/project indicators [Unit]	*Professional engineers / developers* [million]	*Ann. business value* [billion USD/year]	*Quality assurance personnel* [million]	*QA business value* [billion USD/year]	*FI* [%]	*FM* [%]	*Devices* [billion / year]
General IT hardware and devices (incl. personal computers)		2018: 689					2010: 10
Embedded systems (hardware, software, connected embedded devices) in all domains	2014: 1.2	2009: 88 2018: (161)[a]			(35)		2010: 9.8
Industrial embedded systems							2016: 2
Defence, systems, and embedded commercial software engineering					2011: 35		
Embedded software	2014: 0.36	2009: 3.4 2018: (10)	**2010: (0.072)**	**2010: (2.38)**			
General software (overall commercial software engineering, development)	2014: 18.5 2018: 23 2019: 23.9	2013: 407 2018: (515)			(4.6)	2011: 1	

[a]The numbers in parentheses include estimates for 2018 based on the other numbers and corresponding average growth rates.

B Formal Methods in Dependable Systems Engineering

Figure 1 on page 3 depicts our research setting for this survey. There, we describe integrated formal method as an enhancement of formal methods and the assurance of robots and autonomous systemss as a special field of dependable systems engineering. Because many of the discussed issues are of a more general nature, we first prepared a SWOT analysis of FMs in dependable systems engineering in Table 3. Then, we refined this analysis in Table 1 to accommodate specifics of integrated formal methods for robots and autonomous systems.

Table 3: Overview of our general SWOT analysis (according to Piercy and Giles [152]) of "formal methods in dependable systems engineering"

Method Strengths: • Improvement of modelling precision, requirements clarity, verification confidence • High error detection effectiveness, early error removal **Community Strengths:** • Many transfer re-entry points from former case studies with industry • Many dependable systems practitioners perceive FM usefulness as positive	**Method Weaknesses:** • Difficult to learn and apply, many dependable systems practitioners perceive ease of use of FMs as negative • Fragile effectiveness and productivity **Community Weaknesses:** • Lack of compelling evidence of FM effectiveness • Ineffectively communicated in teaching and training
Key Opportunities for FM Transfer and Research: • Desire for early removal of severe errors (Section 4.1) • Desire to learn from accidents and their root causes (Section 4.2) • Desire to be a mature discipline (Section 4.3) • Desire for dependable norms (Section 4.4)	**Method Threats:** • Lack of method scalability • Faulty, tedious, or vacuous proofs • Lack of user education • Poor tool integration, legacy tools and processes **Transfer Threats:** • Lack of researcher engagement in FM transfer • Lack of access to comprehensive high-quality data