

RESEARCH ARTICLE - EMPIRICAL OPEN ACCESS

Reality Check on Formal Methods in Industry: A Study of Verum Dezyne

Michele Chiari¹  | Matteo Camilli² | Marcello M. Bersani²  | Rutger van Beusekom³ | Damian A. Tamburri²¹Institute of Computer Engineering, TU Wien, Vienna, Austria | ²DEIB, Politecnico di Milano, Milan, Italy | ³Verum Software Tools B.V, Eindhoven, The Netherlands**Correspondence:** Michele Chiari (michele.chiari@tuwien.ac.at)**Received:** 27 January 2025 | **Revised:** 9 October 2025 | **Accepted:** 3 November 2025**Keywords:** formal software engineering | formal verification | industry study

ABSTRACT

Many of the classical questions reflecting the actionable use of formal methods in the software industry—“do they scale?” or “are they easily integrated?”—remain without a definitive answer, with many potentially adoptable formal notations being exploited in industry, but in a rather stove-piped and siloed fashion, and with rather few, sometimes anecdotal, success stories to tell. In this article, we strive to provide some more answers to the aforementioned questions on formal methods adoption in industry. We focus our study on a widely adopted formal methods framework in Europe, that is, Verum Dezyne, employed by embedded-computing and hardware-programming companies including Thermo-Fisher, Philips, and more. Results convey a rather interesting story—requiring further study into these matters—but also highlight practical insights for formal practitioners in the field, for example, that formal methods do not disrupt existing processes and scalability issues can be easily addressed by applying mainstream engineering practices, such as decomposition.

1 | Introduction

Despite being an academically active field, formal methods (FM) still struggle to reach widespread adoption in the software industry. The hesitation to embrace FM is often driven by persistent opinions and stereotypes regarding their practical applicability. Commonly cited challenges include an unfavorable learning curve, concerns over cost-effectiveness, difficulties in integrating FM into existing development toolchains, poor scalability, and general skepticism about their usability and return on investment [1, 2]. These concerns, though not always fully substantiated, have nonetheless hindered the diffusion of FM in mainstream software development practices.

In contrast, the electronic design automation (EDA) industry has successfully adopted FM principles early on, incorporating them as integral tools for verifying the correctness of designs,

ensuring robustness, and reducing design errors [3, 4]. EDA companies have leveraged the potential of FM for decades, recognizing their value in optimizing design processes. This stands in stark contrast to the software industry, where the uptake of FM has been for a long time largely confined to niche sectors, such as railway infrastructure [5], aerospace [6], automotive [7], and, to some extent, operating systems [8, 9]. In these areas, the mission- or safety-critical nature of systems makes the costs of failure exceptionally high, driving the need for rigorous verification methods.

The broader software industry, however, has shown limited progress in adopting FM despite the increasing complexity and demands for software quality. The mainstream perception is that FM require highly specialized expertise, which may not be readily available or feasible for general-purpose software development teams. Moreover, FM tools are often

This is an open access article under the terms of the [Creative Commons Attribution](https://creativecommons.org/licenses/by/4.0/) License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2025 The Author(s). Journal of Software: Evolution and Process published by John Wiley & Sons Ltd.

viewed as being incompatible with agile development methodologies and rapid iteration cycles, further discouraging widespread use.

Recently, there has been a notable shift in FM adoption, particularly among large technology companies. Organizations including Microsoft [8], Facebook [10, 11], Huawei [12], and Amazon [13] have integrated FM into their general software development workflows, recognizing its potential to improve software correctness and reliability in the face of increasing complexity. These companies have demonstrated how FM can be adapted to large-scale industrial settings, particularly for systems with critical reliability and security requirements [7]. This trend marks a departure from the historical association of FM with highly specialized sectors and suggests a broader recognition of their value.

Safety-critical systems remain the sector of the software industry with the most consistent interest in FM [14]. Organizations such as NASA's Jet Propulsion Labs [15], which are tasked with developing highly complex, mission-critical software, have turned to FM to ensure high reliability and robustness. In these contexts, the rigorous correctness guarantees provided by FM are indispensable, and the costs associated with FM adoption are justified by the need for ultra-dependable systems. Despite this, FM adoption has, for decades, been limited to a small number of companies, partly due to the sector's specialized requirements and the complexities involved in scaling FM approaches to larger and more diverse software ecosystems.

The recent inclusion of FM in key industry standards, such as DO-178C for aerospace software certification [16] and CENELEC EN 50128 for railway control and protection systems [17], and their recommendation by the US government [18] signal a growing institutionalization of FM in safety-critical domains. These standards recognize the value of FM in verifying that software adheres to strict safety and performance requirements, suggesting that FM are becoming more embedded in critical software development workflows. As FM gain prominence in such high-stakes environments, they may pave the way for broader adoption across other sectors where reliability is becoming increasingly important.

Assessing the evolving role of FM in both safety-critical and broader industrial contexts is essential for understanding the future trajectory of formal verification techniques. Significant questions remain about the state of FM adoption in the wider software industry. What is the current extent of FM integration across different sectors? How do companies adopting FM address the well-documented barriers, such as the challenges of scalability, ease of use, and the steep learning curve? These concerns, along with uncertainty around the return on investment of FM in non-safety-critical applications, continue to shape the adoption landscape. Understanding how companies overcome these obstacles is crucial for charting the future direction of FM in the industry.

In this paper, we explore the state of FM adoption by focusing on the Verum Dezyne framework, a widely used European FM tool. Dezyne provides a model-driven approach to the

development of safety-critical systems, offering verification capabilities that ensure the correctness of the software produced. A key motive for this selection is that Dezyne offers a modern design kit that focuses explicitly on the targets of our study—that is, usability of FM in industrial settings, their impact, and limitations. Despite being employed by several companies, Dezyne still has not undergone any empirical study. In this paper, we strive toward filling this gap. While other valuable substitutes such as Event-B would require a more structured and controlled experimental study, our study is set as an attempt to give context, rather than articulating a comparison among key tools in the same FM sector. The *Guidelines for Performing Empirical Studies on FM* [19] recommend qualitative studies with industrial practitioners as an approach for performing empirical studies on FM. Hence, we conducted qualitative interviews with seven practitioners from companies using Dezyne, including ASML, Thermo-Fisher Scientific, and Philips. The interview covered a range of topics, including the participants' familiarity with FM, the impact of Dezyne's adoption on their development processes, and their perspectives on the future evolution of FM.

Through this study, we aim to answer the following research questions:

RQ1: *What is the degree of integration of FM in industrial processes?*

To answer this, we conducted qualitative interviews with practitioners from companies that have adopted Dezyne in their daily operations. This allows us to understand how deeply FM practices have been embedded in their workflows and the extent of their use in real-world scenarios.

RQ2: *What are the existing challenges to the adoption of FM in the industry?*

Using unstructured data extracted from the interviews, we performed card-sorting to extract a taxonomy of the main challenges. These include issues related to scalability, ease of use, integration with existing toolchains, and the ability to handle complex algorithms during formal verification.

RQ3: *What is the gap between academia and industry regarding FM?*

By comparing the results from our qualitative interviews with existing surveys in the academic literature, we aim to identify the most significant gaps between FM research and its application in industry. This comparison helps us understand why certain academic advancements in FM have not yet translated into widespread industrial adoption.

According to the experience of the companies participating in our study, there is a generally positive impact of FM on the development process, particularly in ensuring software correctness with minimal disruption to ease of development. While scalability issues persist, they can be addressed through modularization. However, challenges remain, particularly in encoding complex algorithms for formal verification. We argue that

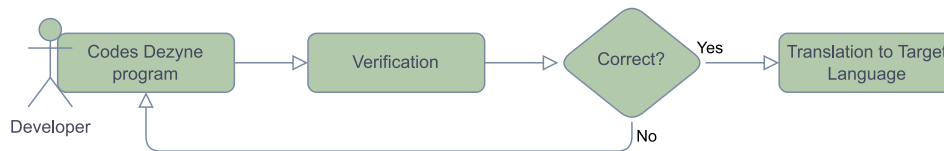


FIGURE 1 | The Dezyne software workflow, an illustration.

addressing these challenges will be crucial for advancing FM adoption in both academia and industry in the coming years.

The paper is structured as follows. In Section 2, we provide an overview of key concepts and background information on Dezyne. In Section 3, we answer the research questions by detailing, for each one of them, our research methodology, and then presenting the key findings emerging from the interviews. In Section 4, we present a roadmap consisting of the main research directions that emerge from our study. In Section 5, we review relevant literature and compare our work with related research. Finally, in Section 6, we summarize our conclusions and suggest directions for future work.

2 | Background: The Dezyne Approach

The Dezyne framework supports the development of formally verified embedded systems through a model-driven approach. It differs from other popular FM approaches in that it is not limited to the analysis of software written in other languages, but it supports the development process by offering a user-friendly language especially targeted to compositional and imperative concurrent programming. This language can be automatically translated to other languages commonly employed in embedded systems, such as C and C++. The Dezyne language has well-defined semantics, enabling formal verification of several desirable properties, such as deadlock freedom, through model checking. A high-level description of a common workflow supported by Dezyne is illustrated in Figure 1.

Dezyne programs consist of components that communicate through formally defined interfaces, following a design-by-contract approach [20, 21]. Each component can implement or use several interfaces, thus making explicit the services it provides and those it requires. Interfaces also (partially) specify the behavior expected from the components implementing them. Components only communicate through messages of two kinds: *In*-events behave like function calls and are synchronous and *out*-events are stored in the message queue of the receiving component and are asynchronous. The behavior specified by interfaces can be implemented in a component through a language that offers variables of several types (including enumerations and bounded integers), assignments, and conditional statements. The language supports functions for code reuse, and iteration through tail-recursive calls. Ultimately, the syntax of the language is close to common general-purpose imperative languages. For this reason, the learning curve of Dezyne is arguably quite flat for fairly experienced programmers.

The semantics of the Dezyne language has been formalized in the *mCRL2* process algebra [22, 23]. The Dezyne verification tool automatically translates it to *mCRL2* process expressions, which

can be automatically and efficiently checked by an optimized model-checking engine [24]. Besides deadlock freedom, Dezyne can check for the reachability of user-defined illegal events, non-observable nondeterminism of components, and noncompliance of components to interfaces, finding bugs that are hard to detect manually or through testing. If a violation is detected, a counterexample trace is provided to the user to facilitate debugging.

The C or C++ code to be run on the target system can be generated automatically from the formally verified specification, thus offering end-to-end correctness guarantees.

2.1 | Interface Example

We illustrate Dezyne through an example that specifies a (simplified) car cruise control system. The system consists of a controller, which communicates with the driver through a human-machine interface (HMI), a monitor for pedals, and the engine throttle. The specification of the HMI interface is in Listing 1, while the specification of the controller component is described in the next section.

Listing 1 Dezyne specification of the HMI of a cruise control.

```

1 interface ihmi {
2   in void enable();
3   in void disable();
4   in void set();
5   out void inactive();
6   behavior {
7     enum State {Disabled,Enabled,Active};
8     State state = State.Disabled;
9     on disable: state = State.Disabled;
10    [!state.Disabled] on enable:
11      /*ignore*/
12    [state.Disabled] on enable:
13      state = State.Enabled;
14    [!state.Enabled] on set: reply (false);
15    [state.Enabled] {
16      on set: {
17        state = State.Active; reply (true);
18      }
19    }
20    [state.Active] {
21      on set: reply (true);
22      on inevitable: {
23        state = State.Enabled; inactive;
24      }
25    }
26  }
27 }
  
```

Listing 2 Dezyne implementation of a cruise control.

```

1 component cruise_control {
2   provides ihmi hmi;
3   requires ipedals pedals;
4   requires ithrottle throttle;
5   requires iassert check;
6   behavior {
7     [hmi.state.Disabled] {
8       on hmi.enable(): pedals.enable();
9       on hmi.disable(): { /* ignore */
10    }
11    [!hmi.state.Disabled] {
12      on hmi.enable(): { /* ignore */
13        on hmi.disable(): {
14          if (throttle.active) throttle.
15            reset();
16          pedals.disable();
17        }
18      }
19      on hmi.set(): {
20        if (hmi.state.Disabled || pedals.
21          engaged)
22          reply (false);
23        else {
24          throttle.setpoint(); reply (true);
25        }
26      }
27      on pedals.engage(), throttle.unset(): {
28        if (hmi.state.Active) {
29          hmi.inactive();
30          if (throttle.active) throttle.
31            reset();
32        }
33      }
34      on pedals.disengage(): { /* ignore */
35      }
36      on check.assert(): {
37        [!hmi.state.Active && throttle.
38          active ||
39          pedals.engaged && throttle.active]
40          illegal;
41        [hmi.state.Active && !pedals.monitor]
42          illegal;
43        [hmi.state.Active && !throttle.
44          active]
45          illegal;
46        [otherwise] {}
47      }
48    }
49  }
50 }

```

The specification of the HMI interface starts by listing messages that the HMI can receive, which correspond to actions that the driver can trigger: The cruise control can be enabled and disabled, and its target speed can be set. The HMI can notify the user whether or not the cruise control is active, by controlling an indicator light through the *inactive()* trigger. The state of the HMI is determined by the enumerated variable state at line 7: It can be *disabled*, *enabled*, or *active*. The meaning of these states is clarified by the specified behavior, given as a list of clauses of the form *[CONDITION] on EVENT*:

{...}, where *CONDITION* is a Boolean expression involving state variables, and *EVENT* is a message that can be received by the interface; if one of these parts is omitted, the case applies to all possible states or messages. In interface implementations, or *components*, the clauses are evaluated from top to bottom, and the first one that applies is taken. Interfaces, however, treat them as if they could occur nondeterministically. Nondeterminism is useful in interfaces because it allows them to *abstract* away details of the implementation that are not relevant for the correctness of the system, thus simplifying the overall system model and speeding up verification. Abstraction is particularly useful when the interface represents an external component written in a different programming language, whose internal state and execution logic are too complex to be entirely represented in Dezyne. On the other hand, components implemented within Dezyne can be verified for determinism, to make sure that they do not present any ambiguous behavior due to overlapping guards.

As we can see, whatever the state of the HMI is, the driver can disable cruise control (line 9). When the HMI is disabled, it can be enabled by the user (line 12). Its state then becomes “*enabled*,” meaning that the controller is ready to conduct the vehicle, but is still idle. When the driver sets the target speed (line 16), the HMI’s state becomes active, and the cruise control is engaged. The reply statement sets the value returned by the trigger, which in this case is a Boolean indicating whether the cruise control was activated successfully. In fact, its activation may sometimes fail, as shown by line 18, which has the same guard as line 16. Whether or not activation succeeds depends on the interface implementation, and here, we use nondeterminism to state that both behaviors are possible. While the interface can be nondeterministic, the implementation is required to be deterministic.

There is also an additional default message, called inevitable (line 24), which describes an event that is guaranteed to occur if no other triggers occur. This means that the HMI can disengage itself at any time (we shall see exactly what this means when implementing the controller).

2.2 | Controller Example

The controller behind the cruise control system is implemented in Listing 2. In Line 2, the “*provides*” directive states that the “*cruise_control*” component implements the *ihmi* interface defined in Listing 1. Then, three “*requires*” directives state that the controller has three *ports* through which it communicates with components implementing the three interfaces *ipedals*, *ithrottle*, and *iassert*. These are, respectively, a sensor monitoring the pedals, the engine throttle PID¹ controller, and a dummy interface used to check some user-specified requirements. These components may be implemented by any means other than Dezyne (e.g., in C or C++), but interfaces just like the one in Listing 1, which we do not show for brevity, allow Dezyne code to interact with them. The compliance of external components to their interfaces is the implementer’s responsibility, and the correctness of cruise control depends on it.

The behavior section, which implements the controller, has the same syntax as in Listing 1. When the cruise control is enabled,

the pedals monitor is also activated (Line 8). Once the driver sets the target speed, the controller checks whether the pedals are being used: This disambiguates the nondeterminism at Lines 16 and 18 of Listing 1. If the pedals are not being pressed (Line 22), the cruise control can be engaged by setting the target speed for the PID controlling the throttle. Recall that, as per Listing 1, the state of the HMI is also set to *Active*. If the pedals are being used while the driver sets the speed (line 20), the trigger fails by returning false, and the cruise control is not activated.

There are two ways in which cruise control can be suspended due to an action by the driver: When the driver disables it through the HMI (line 13), or when they use the pedals (line 25). In both cases, the throttle PID is deactivated. The throttle PID may also become unstable: In this case, it raises an *unset()* event, and the controller deactivates cruise control (line 25). The triggers in line 25 substantiate the “*inevitable*” trigger from Listing 1, Line 24.

Dezyne verifies in a few seconds that the controller is free of deadlocks and unreachable code, and that it will never lead to starvation of interacting clients due to a *livelock*.² Moreover, it proves that *illegal* states cannot be reached, which allows the programmer to specify custom properties through conditional statements. For instance, unreachability of line 35 implies that no unintended acceleration may occur: This is a *safety* requirement whose violation can lead to serious consequences [26]. The two safety requirements that the pedals monitor (line 37) and throttle PID (line 39) are always active when cruise control is active are also proved.

3 | 10 Years of Dezyne in Action: A Reality Check

By investigating Dezyne, we aimed at getting a taste of the extent to which FM are embedded in modern-day software processes or whether such integration manifests into troublesome forms—or *smells*—that could be perceived by their respective practitioners. In the scope of this study, we aim at answering the three research questions anticipated in Section 1. Specifically, we conducted qualitative interviews with practitioners from companies that have adopted Dezyne in their daily operations (Section 3.1). We performed card sorting to extract a taxonomy of the main existing challenges (Section 3.2). Finally, we compared the results from our qualitative interviews with existing surveys in the academic literature to identify the most significant gaps between FM research and its application in industry (Section 3.3).

3.1 | RQ1: What Is the Degree of Integration of Formal Methods in Industrial Processes?

To answer RQ1, we performed a *qualitative survey* including two rounds of interviews with seven employees of companies that embed the Dezyne approach in their software processes. Aiming for high representativeness, we allowed for the selection of professional software engineers that use (or have used) Dezyne in their daily working activities; such selection was entrusted directly to Verum management. Out of an initially selected sample of 15 practitioners in 15 companies, we operated a sample refinement. Specifically, we filtered out and selected the final set of interviewees based on their expertise (3+ years) and experience with FM and Dezyne (6+ months). All practitioners

were >25 y.o. employees involved in large-scale embedded software engineering. In each round of interviews, we had synchronous conversations with one practitioner at a time, in which we asked both closed and open-ended questions. Participants' answers have been recorded and transcribed to carry out qualitative data analysis. Among other peculiarities of Dezyne, the practitioners involved in our study were questioned—using both a Likert scale value and open comments—on (a) the integration of Dezyne into their regular software processes, (b) the general performance and characteristics of the approach and their satisfaction with it, and (c) the general costs of adoption of the approach and its trade-offs. The key results are reported below.

Figure 2 outlines a bubble diagram that captures the major changes that the tool enforces on the regular software lifecycle of the adopting company. The axes of the figure represent the two major dimensions emerging from our analysis: The phase of the software development lifecycle in which the change occurred is on the *x*-axis, and the magnitude of the change brought by Dezyne's adoption on the *y*-axis. Each entry represents a problem or characteristic encountered by the interviewees, and the bubble size is proportional to the number of people that mentioned it. We only report entries that were mentioned by at least two interviewees.

Users report an increase in the quality of code that interacts directly with the hardware (*HW code improvement*), and a decrease in the effort required for finding defects during operation. Clearly, the introduction of Dezyne only causes minor workflow changes in the development and operation phases, but it results in considerably lower efforts spent in more classical defect-finding approaches (e.g., testing, static analyses, code, and component inspection). For example, an interviewee quotation follows:

Before adopting Dezyne, failures were discovered by testing only, and it took a lot of time to get the input to reproduce the error from the testers.

At the same time, perhaps as expected, the largest amount of activities identified as changed by practitioners is concentrated at the design stage. This indicates that some changes might be invisible or latent in development and operations costs, modeling a form of *formality technical debt*, which remains to be explored. For instance, one participant stated:

With Dezyne you have fewer surprises, especially in low-level source code. The main problem is «Am I modelling the right thing?» Otherwise, rubbish in, rubbish out.

Figure 3 captures the perceived expressiveness of the notation with respect to the modeling and abstraction/design difficulty intrinsic to the scenario in which the companies in our study operate. This figure is similar to Figure 2 but tells a considerably different story. The *x*-axis, this time, represents the learnability, or learning curve, of the notation, and the *y*-axis represents the expressiveness of the modeling. Specifically, the notation offered by Dezyne gives practitioners a hard time when expressing complex algorithmic behavior. While many practitioners say that

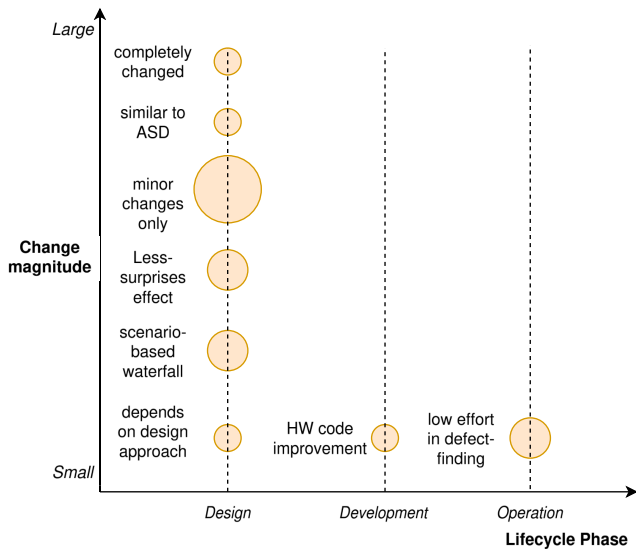


FIGURE 2 | Workflow changes required by the adoption of the approach. The y-axis reports the magnitude of the workflow change caused by the adoption of Dezyne, increasing from the origin. ASD is another product from the company developing Dezyne, which can be considered its predecessor.

it is possible to encode any behavior, all of them remark on the difficulty of use, especially from a learnability perspective. This evidence would lead to conclude that the classical problem of hard-to-use FM is still wide open and deserves further attention. As an example, the following interviewee quotation supports this claim:

Compared to mainstream development processes, Dezyne requires a different approach. It is a bit hard learning it, but when you get the point, it is very useful.

Conversely, as presented in Figure 4, when questioned about the scalability of the approach, few practitioners identify shortcomings. An appreciable 57% of them indeed claim considerable approach scalability. This concern can be even more easily mitigated by applying decomposition, a mainstream engineering practice. A representative quotation follows:

Developing is easier through decomposition. Critical subsystems are built with Dezyne. Most of the time I decompose logically. If components cannot be verified, I have to split them.

In general, verification time can be an obstacle, but it does not prevent successful development. In fact, another developer states:

Up to now, I was able to split components by looking at complexity. So, I was always able to decompose Dezyne models into smaller ones and then verify them in the order of minutes.

This suggests that the scalability of Dezyne and similar formal-verification approaches is indeed a problem not as dire as is

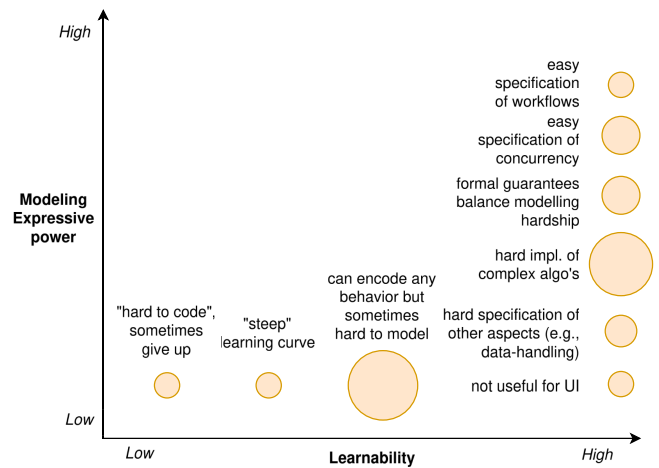


FIGURE 3 | Expressiveness of the notation. The x and y-axes report, respectively, the ease of learning and the expressive power of the Dezyne language, both increasing from the origin.

generally thought. Indeed, Dezyne can be used to encode and verify non-trivial algorithmic behavior when applied separately on each system component. This modularity is enabled by the architecture of the Dezyne framework, in particular thanks to interfaces (see Section 2). Research in FM often overlooks the importance of considering the whole development workflow, and the use of verification in combination with multiple tools in a DevOps fashion deserves further attention.

Summary RQ1: Key findings reveal that the adoption of FM, such as Dezyne, leads to improved hardware-interfacing code quality and reduced efforts in defect detection, particularly during testing and code inspection stages. However, the most significant changes were observed during the design phase, where Dezyne introduced new processes but left the development and operation phases relatively unchanged. Participants highlighted less surprises in low-level code, though they emphasized the importance of correct behavioral models to avoid garbage-in, garbage-out scenarios. While capable of encoding complex behaviors, the steep learning curve posed difficulties for engineers, suggesting that FM may remain hard to use. Scalability is generally considered manageable, especially through decomposition of critical subsystems.

3.2 | RQ2: What Are the Existing Challenges to the Adoption of Formal Methods in the Industry?

In line with our research objective of distilling challenges and opportunities for further research in the direction of industrial-strength FM and their automation, we conducted a card-sorting [27] exercise to prepare a taxonomy of these expected results. Specifically, two of the authors hand-coded [28] the interview materials to find themes and their individual sub-topics—that is, concrete challenges and opportunities emerging from our study—comparing results after an initial coding. Subsequently, we computed the well-known Krippendorff's α coefficient for observation agreement [28]—the α score essentially measures a confidence interval score stemming from the agreement of values

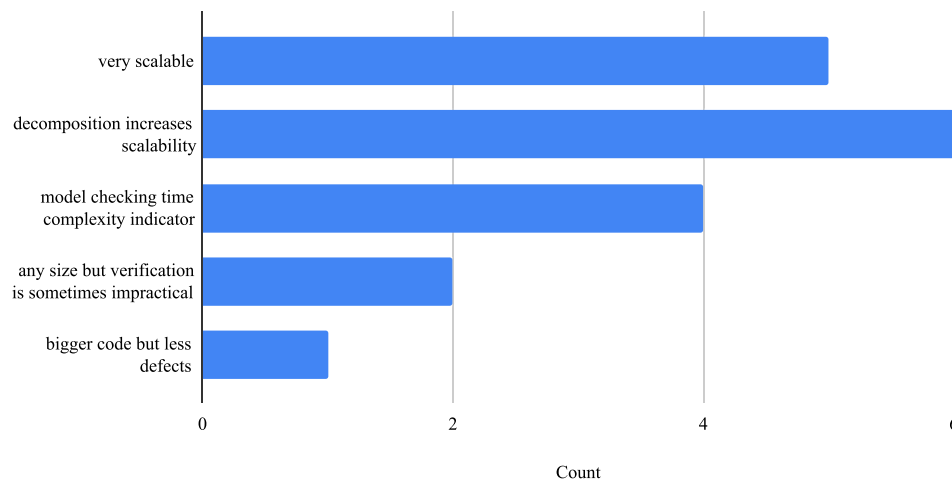


FIGURE 4 | Scalability of the notation and approach in the context of industry-strength systems; the y-axis identifies conclusions drawn by the practitioners concerning the scalability of the Dezyne approach, while the x-axis represents the occurrence frequency, in terms of #companies.

across two distinctly reported observations about the same event or phenomenon. In our case, the value was applied to measure the agreement between the hand-coding results of our analysis. The value featured seven conflicts, which amount to 0.83, hence $\alpha > 0.8$, which is a standard reference value for highly confident observations.

Figure 5 provides a comprehensive outline of the results derived from the aforementioned process. The taxonomy classifies the challenges into five distinct categories:

- *Expressive Power*: This category encompasses challenges ranging from user experience to the complexity of formalisms. It highlights the need for formalisms that are both powerful and user-friendly, balancing intricate technical capabilities with ease of use for practitioners.
- *Workflow Changes*: Here, the challenges reflect how the formalism influences the project pipeline and the extent of mutual impact. It addresses the degree to which adopting a formalism necessitates changes in the existing workflow and how these changes reciprocally affect the formalism's application.
- *Scalability*: Challenges in this category pertain to the adjustments required for a formalism to function effectively in industrial contexts. It includes the need for augmentation and scaling to meet the demands of large-scale projects and real-world applications.
- *Framework Limitations*: This category deals with the integrability of the framework and its no-code characteristics. It focuses on how well the formalism integrates with existing tools and technologies, and the extent to which it supports a no-code approach, making it accessible to non-programmers.
- *Detection Weaknesses*: The final category addresses the types of properties covered by the formalism and their impact on the maintenance and evolution of the system. It highlights weaknesses in detecting certain properties and how these limitations affect the ongoing development and upkeep of the project.

Summary RQ2: Overall, the taxonomy leads to the conclusion that scalability might not be the main challenge of the future concerning FM in industry, as we anticipated in the introduction. Rather—as reflected by equally denser clusters in the taxonomy itself—the challenges concern how the methods are structured internally, and their expressive power. Both these properties seem to reflect concrete challenges that FM researchers could tackle in the immediate future, such as user experience, and the potential mismatch between system complexity and the usability of FM. As systems become more complex, applying FM becomes (sometimes disproportionately) more difficult. Further studies should assess this finding further, perhaps beyond the technologies considered in this study.

3.3 | RQ3: What Is the Gap Between Academia and Industry Regarding Formal Methods?

The 2020 Expert Survey on Formal Methods [29] reports the answers of more than 130 FM experts to 30 questions on the “assessment of FM, FM in research, industry, and education, and the future of FM.” The participants of this survey were picked among people who have a strong relationship with the FMICS conference (e.g., PC chairs, invited speakers) as well as from personal knowledge of the authors. The study does not give a detailed analysis of the participant base, but we can infer from the reported recruitment methodology that most participants have had some academic, possibly post-graduate education in FM. This results in a participant base that is representative of experts who contributed to the development of FM, rather than mere users. In our study, we retrieved opinions from people who are users of FM, but not researchers. Thus, we asked the questions from this survey [29] most related to industry to our participants and compare their views with those of FM experts. Since the size of our sample is small (seven practitioners), we urge readers to take the results of this analysis as purely indicative (cf. Section 3.4). Further studies involving more practitioners are needed to confirm our results with sufficient statistical significance.

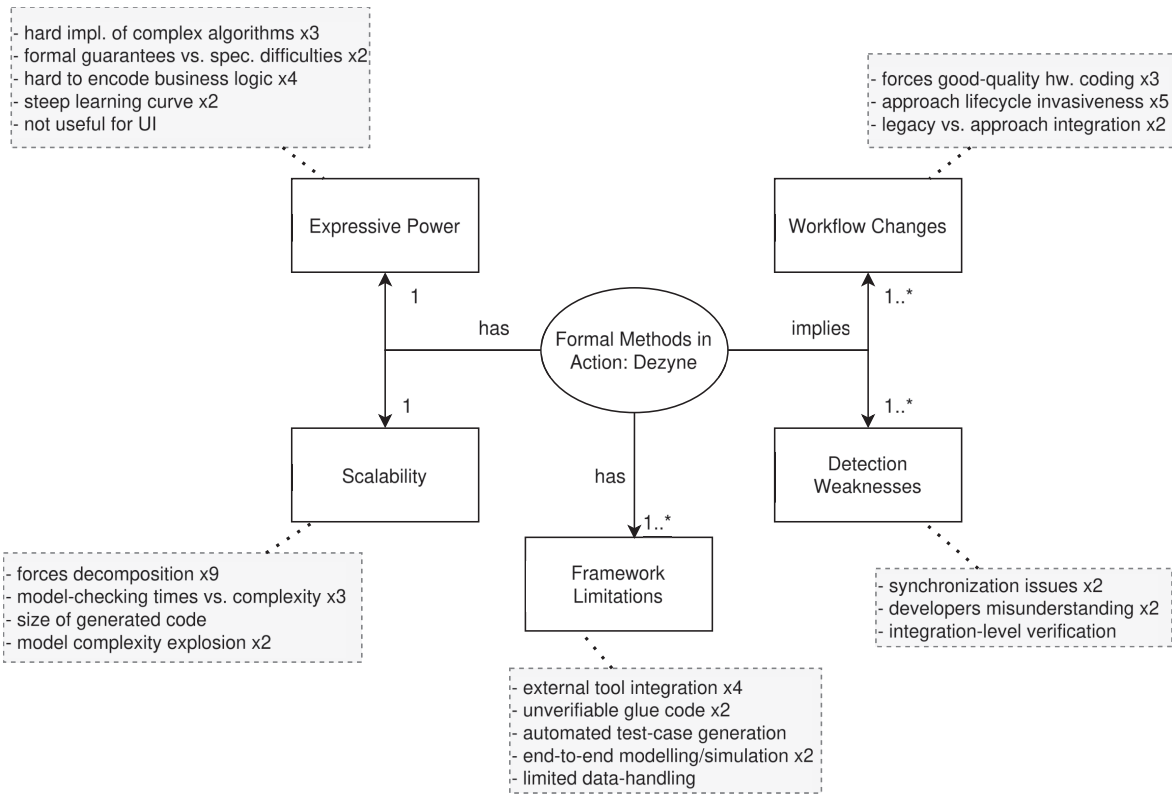


FIGURE 5 | Research challenges and opportunities emerging from our study, a taxonomy; lined boxes identify themes emerging from our card-sorting exercise, while dotted boxes reflect concrete challenges emerging from our study arrows.

The first and most general question is.

“Is it possible to assess the quality of complex (hardware or software) systems without using formal methods?” [29]

We compare our results with those in the Expert Survey in Figure 6. 71% of our respondents answer this question positively, as opposed to only 42% in the Expert Survey. Thus, the majority of the practitioners we interviewed do not believe FM are imperative, but, as we see from the answers to other questions, they believe in their usefulness.

We then asked the participants what the main perceived benefits of FM are, by rating a list of pre-defined areas. The results are reported in Figure 7. Most of the benefit categories received similar scores from the participants of both surveys. The main differences affect the impact of FM on development costs, which our interviewees perceive as more positive and, in particular, their impact on time-to-market, which our participants unanimously recognize as definitely positive.

The next question asks participants to identify phases of the software design life cycle that can benefit from the use of FM. The results are compared in Figure 8. All of our respondents identify “Checking whether models are correct” as the most important task, while they perceive FM less useful in other areas. This might be a result of our participants having experienced exclusively or almost exclusively Dezyne as an FM. Participants in the Expert Survey, on the other hand, were purposefully selected to cover most areas of FM, and they are

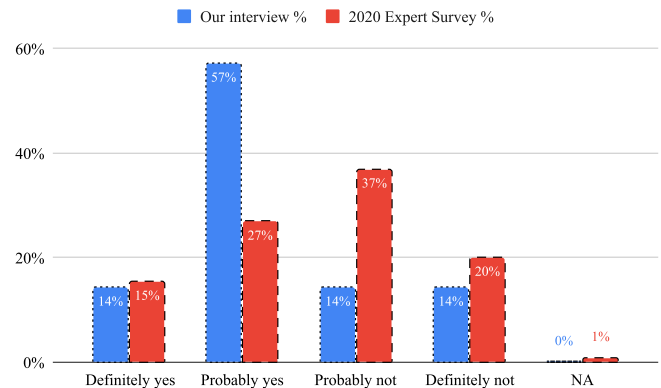


FIGURE 6 | Answers to the question “Is it possible to assess the quality of complex (hardware or software) systems without using formal methods?” taken from Garavel et al. [29].

thus familiar with FM targeting a broader variety of software production phases.

Finally, we asked participants to identify the most urgent areas of improvement on which researchers should focus their work. We report results in Figure 9. Our participant base tends to give a higher priority to more practical features, namely, scalability, applicability, and acceptability. They give less importance to theory and application to new domains. The high importance given to scalability seems to be in contrast with what emerges from Section 3.1—namely, the possibility of overcoming scalability issues through modularization. This apparent contradiction can be explained by the observation that the modularization effort can be time-consuming, and improvements in scalability that make

Summary RQ3: Consistently with existing surveys, our interviewees perceive higher benefits in critical domains like safety-critical systems and cybersecurity. However, our participants felt that FM had a stronger positive impact on development costs and time-to-market. In terms of FM usefulness across the software lifecycle, our respondents focused on “checking whether models are correct” as the primary task for FM, whereas existing surveys also highlight additional stages such as requirements validation and code certification. Priorities for FM development also diverged. Industry practitioners emphasized practical aspects like scalability, applicability, and ease of integration into engineering processes, while existing surveys place more importance on theoretical advancements and exploration of new domains. This reflects the existing gap for needs and focus areas between industry and FM researchers.

the need to split large modules less frequent would be highly beneficial.

3.4 | Threats to Validity

The main threat to *internal validity* concerns the selection of the practitioner sample. Specifically, all members are selected through referrals by Verum management, which may have led to the exclusion of individuals who are less enthusiastic about the approach. While the sample size is relatively small, it is designed to be highly representative of our target population: experts in FM, particularly Verum Dezyne users, with substantial industrial experience in real-world projects.

The main threat to *external validity* arises from the specific practitioners' expertise. All participants are experts in the same FM platform (Verum Dezyne), and their perspectives may not generalize to other FM technologies. Additionally, the business sectors of the companies employing the practitioners may not encompass all possible areas of FM application. Nevertheless, we believe that our results can be considered representative of the phenomena of interest as Verum Dezyne has been adopted by some of the leading

European embedded-computing and hardware-programming companies, such as Philips, ASML, and Thermo Fisher. This threat can be mitigated by comparing our results with surveys involving practitioners using different FM frameworks, which we leave for future work.

Another serious threat to external validity is the small sample size of our survey. This threat is particularly relevant for the answer to RQ3, in which we compare the frequency of answers with another study that involves a much larger sample size [29]. The small sample size prevents the answers to RQ3 from being statistically significant. We thus urge readers to take the results reported in Figures 6–9 as purely indicative.

Limitations of our work highlight the need for further empirical research that involves a larger, more diverse sample, familiar with a broader range of FM technologies, and recruited through more varied channels.

4 | Community Roadmap

The analysis in this paper shows that greater uptake and more widespread use of FM in industry require a dual, yet differentiated, action involving both academia and industry. In this synergistic action, the effectiveness of cooperation is influenced by the remuneration that both parties can obtain. While the industrial need is mostly related to a market placement of a product, and thus of the company itself, and a reduction in costs, the academic need is more characterized by finding financial resources and building theories, techniques and methodologies that influence research. The key to the success of FM, therefore, is not to be found in the discovery of a revolutionary theoretical result but in cooperation between academia and industry that can highlight the value of verification techniques in the development of applications and thus their relevance in both academia and industry. For this reason, we consider the implementation of the following essential actions:

1. Industry must make available to the community the extent of the economic benefits to be gained by applying FM to product development. This requires industry-specific studies that can show how higher product quality can be achieved at

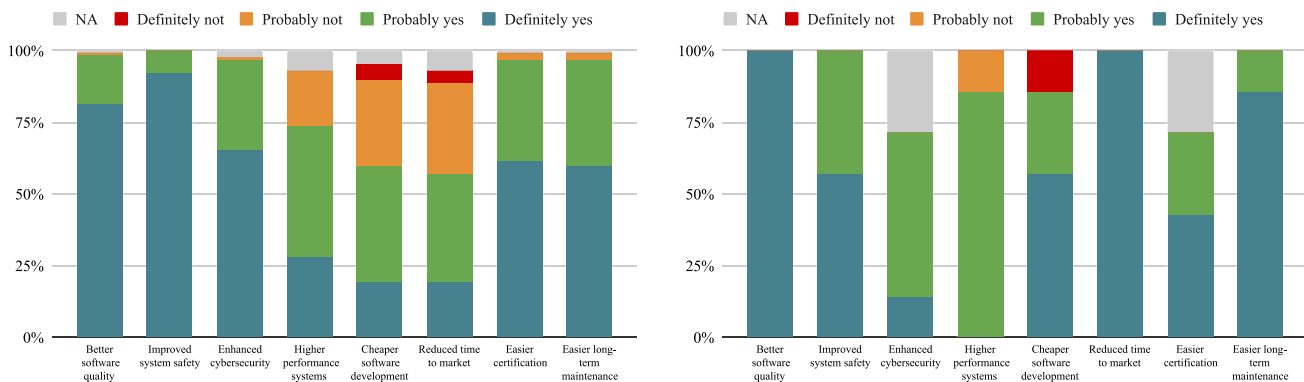


FIGURE 7 | Answers to “Do you believe that formal methods, together with the rigorous use of formal analysis tools, can deliver the promise of:” in Garavel et al. [29] (left) and our interviews (right). Entries on the x-axes are, from left to right, better software quality; improved system safety; enhanced cybersecurity; higher performance systems; cheaper software development; reduced time to market; easier certification; easier long-term maintenance.

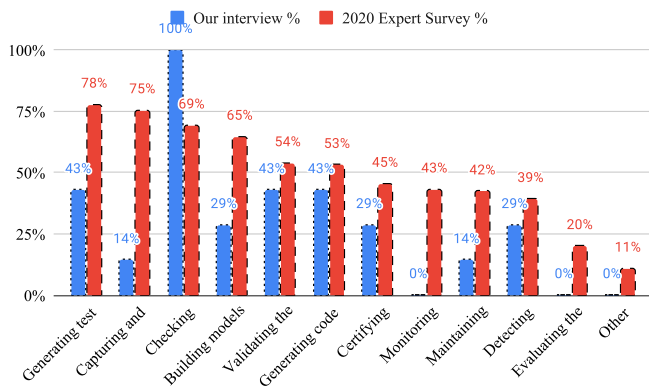


FIGURE 8 | Answers to the question “In which phases of the design life-cycle are formal methods likely to be the most useful?” taken from Garavel et al. [29]. Entries on the x-axis are, from left to right, generating test cases, especially for corner cases; capturing and formalizing requirements; checking whether models are correct; building models of the system; validating the requirements; generating code from models; certifying correctness of the final code; monitoring deployed software at run time; maintaining consistency between models; detecting mistakes in handwritten code; evaluating the test results; other.

lower cost through the use of FM. To support this thesis, it is necessary for academic conferences in the field to enhance industry contributions that show not only the application of a technique to an industrial case but also the economic analysis that demonstrates the magnitude of the benefit obtained.

- Academia must encourage the promotion of FM through the construction of degree programs geared toward training figures such as “FM engineer” and, in general, attempt to include modeling and verification courses in bachelor’s and master’s programs.
- Industry must make itself more open to experimental collaboration with academia. For this, it is essential that industry grasp the importance of projects supported by government agencies that promote collaboration between academic and industrial partners.

A notable example of a funding program of this kind are *Industrial Doctorates*, funded by the European Commission through the “Doctoral Networks” action within the Marie Skłodowska-Curie Actions in the Horizon Europe programme.³

- Academic research needs to be more concerned with what we might call “applicable formal methods,” that is, approaches and methodologies that can be easily employed in industry. Thus, it is necessary to build a body of tools that can be used by industry experts without having specific expertise in formal methods and that can be easily integrated into business production processes. For this reason, academia in synergy with industry must promote the development of spin-offs capable of capturing the needs of companies in order to produce effective, scalable and usable verification tools.

Belli et al. [30] make a preliminary attempt in the direction of Action 1 by presenting a cost–benefit analysis on a small case study on the development of railway communication infrastructure. The analysis quantifies the savings in development costs

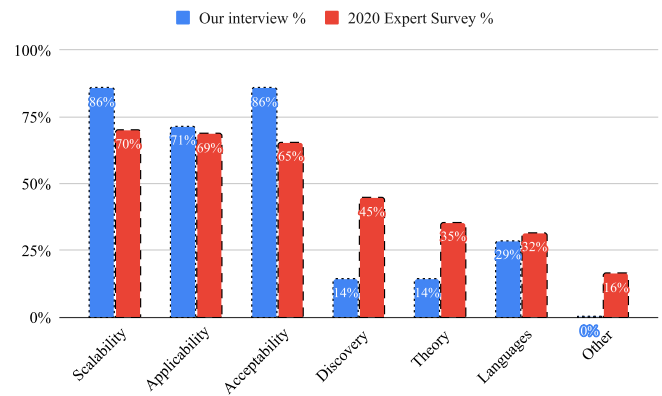


FIGURE 9 | Answers to the question “Which should be the most urgent priorities of researchers working in formal methods?” taken from Garavel et al. [29]. Entries on the x-axis are, from left to right, scalability: design more efficient verification algorithms; applicability: develop more usable software tools; acceptability: enhance integration into software engineering processes; discovery: explore new classes of problems and application domains; theory: search for the next fundamental breakthroughs; languages: design more expressive and user-friendly notations; other. We allowed from 1 to 4 answers.

due to the adoption of FM, and the savings due to the expected reduced frequency of railway service disruptions. However, this work relies on theoretical estimations: We argue for the need for further work reporting data from actual industrial projects.

Action 2 has been recently advocated for by several sources, which highlight the need for increasing the presence of FM in computer science (CS) education, identify issues hindering their teaching, and propose possible solutions [31, 32]. Partly in response to the exclusion of FM as a mandatory knowledge area from the 2023 edition of the ACM/IEEE-CS/AAAI Computer Science Curricula [33] (CS2023), ter Beek et al. [34] provide an overview of several reasons why FM should be integrated in undergraduate curricula as a distinct knowledge area. Each reason is analyzed in detail by separate publications.

Ter Beek et al. [7] show that FM are being increasingly used in industry, including in sectors that are not strictly safety-critical. They point out that, although FM are not included in CS2023 [33] as a distinct subject, eight knowledge areas are related to FM. They advocate for a broader inclusion of FM in both graduate and undergraduate education, observing that “the capacity to abstract and mathematical reasoning that are taught as part of any formal methods course” are “fundamental CS skills that industry would profit from” The worldwide uptake of FM by large companies indeed shows that FM are useful to industry, but the current lack of a systematic FM education in CS professionals hinders a more pervasive industrial uptake of FM.

Broy et al. [35] argue that FM knowledge is beneficial not only to practitioners who employ them, but also to general computer scientists, because *FM thinking* fosters a more thorough way of engineering software systems, by understanding them in depth and with formal precision, with the help of abstractions. Crafting reliable abstractions is essential for any computer engineer, and FM provide the ideal training in this skill. FM are seen as the “engineering discipline of logic,” to which any CS student

should be exposed. The authors thus analyze two ways of integrating FM into CS curricula: as a distinct knowledge area and by augmenting other areas with their respective FM approaches.

Finally, Dongol et al. [36] propose “FM thinking” as a different, more informal and practical way of teaching FM. They split FM thinking into three levels, ranging from informally reasoning about program correctness to “heavy-weight” FM such as model checking and theorem proving. They argue that all CS students should be exposed to the lower levels, while only students who are interested in a career in FM should be taught heavy-weight FM. They further highlight the need to instruct CS teachers on how to embed FM thinking in general CS courses, such as introductory programming courses, and show how to do so within existing CS2023 [33] knowledge areas.

In their Manifesto for Applicable FM, Gleirscher et al. [37] provide 10 principles to guide researchers in developing FM that are more attractive for industry. Many of these principles agree with the aims of action 4 and the findings of this paper: FM should have a clear methodology for tackling the complexity of real-world systems (such as modularity support for scalability), provide integration with widely used technologies employed in software development, support users through mature and easy-to-use toolsets, and be applicable to industry-scale systems. Moreover, extensive documentation and teaching materials should allow average graduates and engineers to learn how to use a FM tool with reasonable effort, and the benefits of employing FM should be convincingly demonstrated through empirical evidence, such as case studies or controlled experiments.

5 | Related Work

The results presented in this paper relate both to studies that assess the adoption of FM in industry and to those that survey opinions of researchers and practitioners.

5.1 | Industrial Adoption of FM

Several studies have reported on industrial adoption of FM. One of the earliest ones [38] surveys the state-of-the-art of FM in 1996 and gives an account of several success stories of FM adoption in public and private industry, including those arising from collaboration with academia.

A more recent (2009) study by Woodcock et al. [6] surveys 62 cases of industrial application of FM, giving a more detailed description of eight *highlighted projects*. Additionally, the authors administered a questionnaire to practitioners involved in 56 of the projects, including some questions overlapping with our study. The answers create a picture similar to what we report: Many of the participants stated that the use of FM improved the overall quality of the software and helped decrease development time and costs, because the initial increases caused by FM were more than compensated by the decreased need for testing and fixing bugs.

Many studies cover successful applications of FM arising from collaboration between industry and academia. A book edited by

S. Gnesi and T. Margaria [39] collects introductory chapters on several successful verification frameworks that have originated in academia and were employed within safety-critical industrial projects. Some of these frameworks were developed and maintained by independent companies.

Ferrari and ter Beek [40] conducted a systematic mapping study of FM literature applied to railway systems, gathering more than 300 papers. Only 32% of the studies involve industry partners, and the authors found that “many articles present only examples (41%) or experience reports (38%)” that have a limited empirical relevance. The article also includes a thorough account of previous surveys on FM, to which we refer the interested reader, together with the 2009 study by Woodcock [6], especially concerning works published in the ’90s.

Kulik et al. [41] present a survey of FM applications in cyber security research, collecting 115 works published within the previous 10 years and categorizing them according to a taxonomy of FM application case studies based on the type of FM employed, the industrial domain, and the level of abstraction at which verification is performed. The taxonomy aims at helping researchers and practitioners in orienting themselves among available FM techniques, providing a reference for matching the right tool to the problem at hand. They identify a trend of increasing FM employment for securing financial applications, fostered by the rise of cryptocurrencies. They find that model checking is the most popular technique among practitioners in part due to better tool support, but argue that it may fail to scale up to large systems, for which theorem proving is better suited. However, theorem proving still has a steep learning curve and worse tool support, and an improvement in its accessibility is needed.

The German Federal Office for Information Security commissioned a report on the status of FM [14], in which several taxonomies of FM are reported, based on, among other things, application domains, design flows, and methodologies. The report also observes how the success of FM has remained mostly secluded to the niche of safety- and life-critical systems and highlights the worrying issue that, in part due to this limited user base, several industrial developers of FM are failing to secure their maintenance, leading to their disappearance (one of these is Esterel Studio [42], also described in [39]).

Ter Beek et al. [7] give an overview of the current offer of FM frameworks and tools, with particular attention to their industrial applicability, and survey the industrial adoption of FM, attesting their employment in the following sectors: railways, automotive, aerospace, operating systems, cloud services and security, hardware design, lithography manufacturing, and mobile devices. The study confirms the high prevalence of FM in safety-critical sectors and EDA but also shows a trend of increasing uptake in other sectors that are not safety critical but are *mission critical*, and seek to prevent the considerable financial losses caused by software failures.

5.2 | Opinions of Researchers and Practitioners

Several studies surveying users’ opinions on FM have been conducted in the last few years. The previously mentioned study by

Woodcock et al. [6] is one of the largest. Moreover, we already analyzed and compared with the 2020 Expert Survey on Formal Methods [29] in Section 3.3.

In 2001, Snook and Harrison [43] conducted five structured interviews of FM practitioners. The most used frameworks were the Z notation and the B method. The Z notation is a formal specification language based on axiomatic set theory, lambda calculus, and first-order logic [44]. The B method is a formal language based on an abstract machine notation that allows for building a system implementation through multiple refinement steps starting from the initial, high-level specification [45]. Although this work reflects a rather outdated situation (from the early 2000s), many of its results agree with ours. In both studies, practitioners claim that the use of FM in early design phases significantly reduces efforts required by other quality assurance activities such as testing; the most difficult activity in using FM is properly modeling the system with the right abstractions; learning formal specification languages is generally not an issue for developers, although the learning curve may be steep; scalability issues can be tamed by component decomposition and encapsulation; tool support, applicability, and acceptability are major factors for a company when choosing whether to employ FM.

A survey on the use of FM in the railway industry has been conducted by Basile et al. [46] on the participants of the RSSRail'17 conference, obtaining a sample of 44 respondents evenly divided between industry and academia. The answers show that industrial practitioners had mostly employed tools related to the B method, followed by MATLAB (e.g., Simulink, which is a graphical model-based tool for the simulation of dynamical systems) and SCADE. None of these tools offers support for automated verification. Although the semantics of Simulink are not rigorously formalized [47], Simulink models can be translated to hybrid automata, whose verification is undecidable for many properties of interest [48, 49]. This survey, however, confirms that the B method is an established technique in the railway industry. Many of the tools on which academic research is currently most active—primarily model checking tools—are only mentioned by academics, and very few to no practitioners. The most desirable features for a FM tool are found to be the ability to perform formal verification, modeling, and simulation. Code generation is also mentioned by 22% of the respondents. The most relevant nonfunctional features are maturity, ease of learning, quality of documentation, and ease of integration into the CENELEC EN 50128 process [17].

Ter Beek et al. [50] summarize the results of two more questionnaires answered by populations mostly consisting of practitioners from the railway industry and compare them with Basile et al. [46], showing that they reach substantially similar conclusions.

A study on the barriers that hinder the adoption of FM in the aerospace industry has been conducted by Davis et al. [2] by interviewing 31 practitioners from the aerospace sector. The main barrier categories they identified concern training of the workforce, tool quality, user-friendliness and support, and specific features of the industrial environment.

The largest empirical study of FM we know of has been conducted in 2019 by Gleirscher and Marmsoler [51] through an online survey on a sample of 216 people (response rate 1%–2%) of which 31% are pure academics, only 11% are pure practitioners, and the rest have a mixed background. The study reports statistics on several topics, including the distribution of FM techniques used, industrial fields, and roles of respondents. The toughest challenges for the adoption of FM are reported to be scalability and skills and education. Tool deficiency is also listed as an obstacle to FM adoption.

We note that most of the studies we reported above are only based on online surveys gathering very heterogeneous samples, often comprising individuals with an academic—rather than industrial—background. Our study is instead carried out on a carefully selected sample, whose members have exclusively industrial experience in the use of the same FM framework, Dezyne, and data have been gathered through structured online interviews, rather than surveys.

5.3 | Tool Comparisons

A few works compare available FM tools from the point of view of practical applicability, mainly in the railway domain.

Haxthausen et al. [52] compare two different approaches developed specifically for railway interlocking systems. Mazzanti et al. [53] compare models of a railway interlocking algorithm and verification results obtained with 7 tools (UMC, NuSMV, Promela/SPIN, mCRL2, FDR4, CPN Tools, and CADP). The ABZ'18 conference included a case study on modeling of a standardized European railway interlocking system with different tools [54].

Ferrari et al. [55] do a more systematic comparison of nine FM tools, some of them with a predominantly industrial adoption (Atelier B, ProB, Simulink), and others more related to academia (NuSMV, SPIN, UPPAAL). Three of the authors employed different tools to model a railway control system, and the other two analyzed their experience reports; the resulting data was assessed by 12 external experts. The study categorizes tools based on their adequacy for development phases among *requirements*, *high-level design*, *detailed design*, and *implementation*; purposes among *system modeling*, *simulation*, *quantitative analysis*, and *formal verification*; system types among *single system*, *composite system*, and *system-of-systems*. Moreover, they highlight the main issues that they encountered with each tool. The results show that few tools are well suited for all development phases, and many tools present acceptability issues such as scarce interoperability and integration with other tools that are better at addressing different purposes, poor user interfaces, and require the involvement of experts in different domains and techniques. In this classification, Verum Dezyne is particularly suited for high-level and detailed design, and implementation (thanks to automated code generation), and it supports system modeling, simulation, and formal verification through model checking; its models are systems-of-systems—that is, “the composition of multiple instances of single systems” [55]. Dezyne is distributed together with an integrated development environment

(IDE), which offers an advanced user interface, and its ability to target different programming languages facilitates its integration in heterogeneous systems.

Ferrari et al. [56] provide a structured comparison between 13 tools (including nine from the previous study [55]) applied to the early design phase of a railway signaling system. The authors

elicit a list of features of interest and evaluate the tools based on them. We report in Table 1 our evaluation of Verum Dezyne according to the same features. The usability of seven tools is evaluated by 12 railway experts, mostly from industry after viewing a demonstration of their usage for modeling a railway system. The main resulting insights are that usability is acceptable for most tools, and tools are generally mature enough; the

TABLE 1 | Analysis of Verum Dezyne according to the features collected by Ferrari et al. [56].

Category	Feature	Dezyne
Development Functionalities	Specification/modeling	Textual (TEXT)
	Code generation	Yes
	Documentation and report generation	Partial (diagrams)
	Requirements traceability	No
	Project management	Yes
Verification functionalities	Simulation	Graphical and textual (MIX)
	Formal verification	Linear-time Model Checking
	Large-scale verification technique	On-the-fly, symbolic, compositionality
	Model-based testing	No
Language expressiveness	Non-determinism	External, internal
	Concurrency	(a)synchronous
	Timing aspects	No
	Stochastic or probabilistic aspects	No
	Language modularity	High
	Supported data structures	Basic
	Float support	No
Tool flexibility	Backward compatibility	Yes
	Standard input format	Open
	Import from or export to other tools	Low
	Tool Modularity	High
	Team support	Yes
Maturity	Industrial diffusion	High
	Development stage	Mature
Usability	Availability of customer support	Yes
	Graphical user interface	Yes
	Mathematical background	Basic
	Quality of documentation	Excellent
Company constraints	Cost	Free
	Supported platforms	Windows, Linux
	Complexity of license management	Easy
	Easy to install	Yes
Domain-specific criteria	CENELEC certification	No
	Integration in the CENELEC process	Not applicable

main deficiencies concern “lack [of] support for development features and process-integration aspects,” and scarce portability due to “non-standard languages and specialized verification capabilities.”

6 | Conclusion

The present work sought to look into the state of FM penetration in industry, which, however considerable, is still underinvestigated from the perspective of adoption, and whose limitations are clear only to long-standing practitioners or seasoned FM researchers. To obtain these results, we focused our lens of analysis onto users of an exponent of FM in industry, namely, the Verum Dezyne tool-suite.

Rotating around this tool, we conducted an empirical analysis of the practice and experiences stemming from the practitioners that consider the tool a critical technology for their product space. We tried to answer three research questions concerning, respectively, the degree of integration of FM in industrial processes, the existing challenges to FM adoption in industry, and the opinion gap between industry and academia.

While our study focuses on a specific approach within one organizational context and is not intended to support broad generalization, it provides a focused, empirical account of the practical benefits and limitations encountered when applying FMs in a real-world setting. Our findings highlight several mature aspects of FM adoption in industry, as well as specific strengths of the Dezyne approach. For example, Dezyne is able to incrementally and iteratively address monolith decomposition through modularization, a feature that is crucial for its industrial success. We found five different categories of challenges hindering industrial adoption of FM and classified each one of them in a taxonomy. According to the practitioners we interviewed, the most pressing issues concern scalability, ease of use, and integration into existing software engineering processes. The latter, in particular, is a limitation too often underestimated and ignored by the academic FM community. Thus, our study reflects the need for further work in this direction by the technology owners.

We plan to compare our findings with studies involving practitioners using other FM frameworks adopted in industry. Such comparative analyses would help validate the generality of the observed challenges and guide the development of FM tools and practices.

Acknowledgments

The authors would like to thank all the practitioners that supported this work as well as Bert de Jonge, the CEO of Verum Software Tools B.V., for their valuable collaboration in the scope of this work. The authors also thank the anonymous referees for their valuable comments and advice for improving the paper. Open Access funding provided by Technische Universität Wien/KEMÖ.

Data Availability Statement

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

Endnotes

- ¹ A proportional–integral–derivative (PID) controller is an automated mechanism that uses a feedback loop to keep a system variable close to a given setpoint [25].
- ² A livelock occurs when a component is permanently busy with internal behavior and fails to interact with other components or the external environment (i.e., we check for a *liveness* requirement).
- ³ <https://marie-sklodowska-curie-actions.ec.europa.eu/>

References

1. D. Bjørner and K. Havelund, “40 Years of Formal Methods—Some Obstacles and Some Possibilities?” in *Proc. FM’14, LNCS*, vol. 8442, eds. C. B. Jones, P. Pihlajasaari, and J. Sun (Springer, 2014), 42–61, https://doi.org/10.1007/978-3-319-06410-9_4.
2. J. A. Davis, et al., “Study on the Barriers to the Industrial Adoption of Formal Methods,” in *Proc. FMICS 2013, LNCS*, vol. 8187, eds. C. Pecheur and M. Dierkes (Springer, 2013), 63–77, https://doi.org/10.1007/978-3-642-41010-9_5.
3. L. Fix, “Fifteen Years of Formal Property Verification in Intel,” in *25 Years of Model Checking—History, Achievements, Perspectives, LNCS*, vol. 5000, eds. O. Grumberg and H. Veith (Springer, 2008), 139–144, https://doi.org/10.1007/978-3-540-69850-0_8.
4. D. S. Hardin, ed., *Design and Verification of Microprocessor Systems for High-Assurance Applications* (Springer, 2010), <https://doi.org/10.1007/978-1-4419-1539-9>.
5. M. J. Butler, P. Körner, S. Krings, et al., “The First Twenty-Five Years of Industrial Use of the B-Method,” in *Proc. FMICS’20, LNCS*, vol. 12327, eds. M. H. ter Beek and D. Nickovic (Springer, 2020), 189–209, https://doi.org/10.1007/978-3-030-58298-2_8.
6. J. Woodcock, P. G. Larsen, J. Bicarregui, and J. S. Fitzgerald, “Formal Methods: Practice and Experience,” *ACM Computing Surveys* 41, no. 4 (2009): 1–36, <https://doi.org/10.1145/1592434.1592436>.
7. M. H. ter Beek, R. Chapman, R. Cleaveland, et al., “Formal Methods in Industry,” *Formal Aspects of Computing* 37, no. 1 (2025): 1–38, <https://doi.org/10.1145/3689374>.
8. T. Ball, V. Levin, and S. K. Rajamani, “A Decade of Software Model Checking With SLAM,” *Communications of the ACM* 54, no. 7 (2011): 68–76, <https://doi.org/10.1145/1965724.1965743>.
9. J. Lawall and G. Muller, “Automating Program Transformation With Coccinelle,” in *NFM’22, LNCS*, vol. 13260, eds. J. V. Deshmukh, K. Havelund, and I. Perez (Springer, 2022), 71–87, https://doi.org/10.1007/978-3-031-06773-0_4.
10. C. Calcagno, et al., “Moving Fast With Software Verification,” in *Proc. NFM 2015, LNCS*, vol. 9058, eds. K. Havelund, G. J. Holzmann, and R. Joshi (NASA, Springer, 2015), 3–11, https://doi.org/10.1007/978-3-319-17524-9_1.
11. D. Distefano, M. Fähndrich, F. Logozzo, and P. W. O’Hearn, “Scaling Static Analyses at Facebook,” *Communications of the ACM* 62, no. 8 (2019): 62–70, <https://doi.org/10.1145/3338112>.
12. S. Gao, B. Zhan, D. Liu, et al., “Formal Verification of Consensus in the Taurus Distributed Database,” in *FM’21, LNCS*, vol. 13047, eds. M. Huisman, C. S. Pasareanu, and N. Zhan (Springer, 2021), 741–751, https://doi.org/10.1007/978-3-030-90870-6_42.
13. C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, “How Amazon Web Services Uses Formal Methods,” *Communications of the ACM* 58, no. 4 (2015): 66–73, <https://doi.org/10.1145/2699417>.
14. H. Garavel and S. Graf. “Formal Methods for Safe and Secure Computers Systems, BSI Study 875, Federal Office for Information Security, Bonn, Germany,” (2013), <https://www.bsi.bund.de/SharedDocs/Downl>

oads/DE/BSI/Publikationen/Studien/formal_methods_study_875/formal_methods_study_875.html.

15. J. Kelly and R. Covington, “Experience With Formal Methods Techniques at the jet Propulsion Laboratory From a Quality Assurance Perspective, 9th Computing in Aerospace Conference, American Institute of Aeronautics and Astronautics,” (1993), <https://doi.org/10.2514/6.1993-4518>.
16. B. M. Brosgol, *DO-178C: The Next Avionics Safety Standard*, eds. R. E. Sward, et al. (ACM, 2011), 5–6, <https://doi.org/10.1145/2070337.2070341>. Proc. SIGAda’11
17. European Committee for Electrotechnical Standardization, “CENELEC - EN 50128: Railway Applications—Communication, Signalling and Processing Systems—Software for Railway Control and Protection Systems,” <https://standards.globalspec.com/std/14317747/en-50128> (2020).
18. White House Office of the National Cyber Director (ONCD), “Back to the Building Blocks: A Path Toward Secure and Measurable Software, NA,” The White House, Washington D.C., USA, (2024), <https://bidenwhitehouse.archives.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf>.
19. M. H. ter Beek and A. Ferrari, “Empirical Formal Methods: Guidelines for Performing Empirical Studies on Formal Methods,” *Software* 1, no. 4 (2022): 381–416, <https://doi.org/10.3390/software1040017>.
20. B. Meyer, “Design by Contract,” in *Object-Oriented Software Construction, 2nd Edition*, ed. B. Meyer (Prentice-Hall, 1997), <https://bertrandmeyer.com/OOSC/2/>.
21. C. Silva, S. Guérin, R. Mazo, and J. Champeau, *Contract-Based Design Patterns: A Design by Contract Approach to Specify Security Patterns*, eds. M. Volkamer and C. Wressnegger (ARES, ACM, 2020), 1–9.
22. J. F. Groote and M. R. Mousavi, *Modeling and Analysis of Communicating Systems* (MIT Press, 2014), <https://doi.org/10.7551/mitpress/9946.001.0001>.
23. R. van Beusekom, J. F. Groote, P. F. Hoogendijk, et al., “Formalising the Dezyne Modelling Language in mCRL2,” in *FMICS-AVoCS’17, LNCS*, vol. 10471, eds. L. Petrucci, C. Seceleanu, and A. Cavalcanti (Springer, 2017), 217–233, https://doi.org/10.1007/978-3-319-67113-0_14.
24. O. Bunte, J. F. Groote, J. J. A. Keiren, et al., “The mCRL2 Toolset for Analysing Concurrent Systems—Improvements in Expressivity and Usability,” in *TACAS 2019, LNCS*, vol. 11428, eds. T. Vojnar and L. Zhang (Springer, 2019), 21–39, https://doi.org/10.1007/978-3-030-17465-1_2.
25. M. King, “PID Algorithm,” in *Process Control: A Practical Approach* (John Wiley & Sons, 2011), 29–89, <https://doi.org/10.1002/9780470976562.ch3>.
26. S. Edelstein, “2018-2023 Nissan Leaf EV Recalled for Cruise-Control Acceleration Flaw, Green Car Reports,” (2023), https://www.greencarreports.com/news/1140309_2018-2023-nissan-leaf-ev-recalled-for-cruise-control-acceleration-flaw.
27. W. Hudson, “Old Cards, New Tricks: Applied Techniques in Card Sorting,” in *Proc. BCS HCI 2007*, eds. T. C. Ormerod and C. Sas (BCS, 2007), 225–226.
28. K. Krippendorff, *Content Analysis: An Introduction to Its Methodology*, second ed. (Sage Publications, 2004).
29. H. Garavel, M. H. ter Beek, and J. van de Pol, “The 2020 Expert Survey on Formal Methods,” in *Proc. FMICS 2020, LNCS*, vol. 12327, eds. M. H. ter Beek and D. Nickovic (Springer, 2020), 3–69, https://doi.org/10.1007/978-3-030-58298-2_1.
30. D. Belli, A. Fantechi, S. Gnesi, et al., “The 4SECURail Case Study on Rigorous Standard Interface Specifications,” in *Proc. FMICS’23, LNCS*, vol. 14290, eds. A. Cimatti and L. Titolo (Springer, 2023), 22–39, https://doi.org/10.1007/978-3-031-43681-9_2.
31. A. Cerone, M. Roggenbach, J. Davenport, et al., “Rooting Formal Methods Within Higher Education Curricula for Computer Science and Software Engineering—A White Paper,” in *Proc. FMFun’19, Communications in Computer and Information Science*, vol. 1301, eds. A. Cerone and M. Roggenbach (Springer, 2019), 1–26, https://doi.org/10.1007/978-3-030-71374-4_1.
32. S. Ramnath and S. Walk, “Structuring Formal Methods Into the Undergraduate Computer Science Curriculum,” in *Proc. NFM’24, LNCS*, vol. 14627, eds. N. Benz, D. Gopinath, and N. Shi (Springer, 2024), 399–405, https://doi.org/10.1007/978-3-031-60698-4_24.
33. A. N. Kumar, R. K. Raj, S. G. Aly, et al., “Computer Science Curricula 2023, ACM/IEEE-Cs/AAAI,” (2024), <https://doi.org/10.1145/3664191>.
34. M. H. ter Beek, M. Broy, and B. Dongol, “The Role of Formal Methods in Computer Science Education,” *ACM Inroads* 15, no. 4 (2024): 58–66, <https://doi.org/10.1145/3702231>.
35. M. Broy, A. D. Brucker, A. Fantechi, et al., “Does Every Computer Scientist Need to Know Formal Methods?,” *Formal Aspects of Computing* 37, no. 1 (2025): 1–17, <https://doi.org/10.1145/3670795>.
36. B. Dongol, C. Dubois, S. Hallerstede, et al., “On Formal Methods Thinking in Computer Science Education,” *Formal Aspects of Computing* 37, no. 1 (2025): 1–23, <https://doi.org/10.1145/3670419>.
37. M. Gleirscher, J. van de Pol, and J. Woodcock, “A Manifesto for Applicable Formal Methods,” *Software and Systems Modeling* 22, no. 6 (2023): 1737–1749, <https://doi.org/10.1007/S10270-023-01124-2>.
38. E. M. Clarke and J. M. Wing, “Formal Methods: State of the Art and Future Directions,” *ACM Computing Surveys* 28, no. 4 (1996): 626–643, <https://doi.org/10.1145/242223.242257>.
39. S. Gnesi and T. Margaria, eds., *Formal Methods for Industrial Critical Systems: A Survey of Applications* (Wiley, 2012), <https://doi.org/10.1002/9781118459898>.
40. A. Ferrari and M. H. ter Beek, “Formal Methods in Railways: A Systematic Mapping Study,” *ACM Computing Surveys* 55, no. 4 (2023): 69:1–69:37, <https://doi.org/10.1145/3520480>.
41. T. Kulik, B. Dongol, P. G. Larsen, et al., “A Survey of Practical Formal Methods for Security,” *Formal Aspects of Computing* 34, no. 1 (2022): 1–39, <https://doi.org/10.1145/3522582>.
42. G. Berry, “Esterel v7: From Verified Formal Specification to Efficient Industrial Designs,” in *Proc. FASE’05, LNCS*, vol. 3442, ed. M. Cerioli (Springer, 2005), 1, https://doi.org/10.1007/978-3-540-31984-9_1.
43. C. F. Snook and R. Harrison, “Practitioners’ Views on the Use of Formal Methods: An Industrial Survey by Structured Interview,” *Information and Software Technology* 43, no. 4 (2001): 275–283, [https://doi.org/10.1016/S0950-5849\(00\)00166-X](https://doi.org/10.1016/S0950-5849(00)00166-X).
44. J. M. Spivey, *Z Notation—A Reference Manual*, 2nd ed. (Prentice Hall, 1992).
45. D. Cansell and D. Méry, “Foundations of the B Method,” *Computers and Artificial Intelligence* 22, no. 3–4 (2003): 221–256, <http://www.cai.sk/ojs/index.php/cai/article/view/456>.
46. D. Basile, et al., “On the Industrial Uptake of Formal Methods in the Railway Domain—A Survey With Stakeholders,” in *Proc. IFM’18, LNCS*, vol. 11023, eds. C. A. Furia and K. Winter (Springer, 2018), 20–29, https://doi.org/10.1007/978-3-319-98938-9_2.
47. R. Alur, Formal Verification of Hybrid Systems, Chakraborty, S., A. Jerraya, S. K. Baruah, and S. Fischmeister (eds.), *Proc. EMSOFT’11*, ACM, 2011, 273–278, <https://doi.org/10.1145/2038642.2038685>.
48. R. Alur, C. Courcoubetis, N. Halbwachs, et al., “The Algorithmic Analysis of Hybrid Systems,” *Theoretical Computer Science* 138, no. 1 (1995): 3–34, [https://doi.org/10.1016/0304-3975\(94\)00202-T](https://doi.org/10.1016/0304-3975(94)00202-T).
49. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, “What’s Decidable About Hybrid Automata?,” *Journal of Computer and System Sciences* 57, no. 1 (1998): 94–124, <https://doi.org/10.1006/JCSS.1998.1581>.

50. M. H. ter Beek, A. Borälv, A. Fantechi, et al., “Adopting Formal Methods in an Industrial Setting: The Railways Case,” in *Proc. FM 2019, LNCS*, vol. 11800, eds. M. H. ter Beek, A. McIver, and J. N. Oliveira (Springer, 2019), 762–772, https://doi.org/10.1007/978-3-030-30942-8_46.
51. M. Gleirscher and D. Marmosler, “Formal Methods in Dependable Systems Engineering: A Survey of Professionals From Europe and North America,” *Empirical Software Engineering* 25, no. 6 (2020): 4473–4546, <https://doi.org/10.1007/S10664-020-09836-5>.
52. A. E. Haxthausen, H. N. Nguyen, and M. Roggenbach, “Comparing Formal Verification Approaches of Interlocking Systems,” in *Proc. RSSRail’16, LNCS*, vol. 9707, eds. T. Lecomte, R. Pinger, and A. B. Romanovsky (Springer, 2016), 160–177, https://doi.org/10.1007/978-3-319-33951-1_12.
53. F. Mazzanti, A. Ferrari, and G. O. Spagnolo, “Towards Formal Methods Diversity in Railways: An Experience Report With Seven Frameworks,” *International Journal on Software Tools for Technology Transfer* 20, no. 3 (2018): 263–288, <https://doi.org/10.1007/S10009-018-0488-3>.
54. M. J. Butler, T. S. Hoang, A. Raschke, and K. Reichl, “Introduction to Special Section on the ABZ 2018 Case Study: Hybrid ERTMS/ETCS Level 3,” *International Journal on Software Tools for Technology Transfer* 22, no. 3 (2020): 249–255, <https://doi.org/10.1007/S10009-020-00562-3>.
55. A. Ferrari, F. Mazzanti, D. Basile, M. H. ter Beek, and A. Fantechi, *Comparing Formal Tools for System Design: A Judgment Study*, eds. G. Rothermel and D. Bae (ACM, 2020), 62–74. Proc. ICSE’20
56. A. Ferrari, F. Mazzanti, D. Basile, and M. H. ter Beek, “Systematic Evaluation and Usability Analysis of Formal Methods Tools for Railway Signaling System Design,” *IEEE Transactions on Software Engineering* 48, no. 11 (2022): 4675–4691, <https://doi.org/10.1109/TSE.2021.3124677>.