

Noname manuscript No. (will be inserted by the editor)
--

A User Study for Evaluation of Formal Verification Results and their Explanation at Bosch

Arut Prakash Kaleeswaran · Arne Nordmann · Thomas Vogel · Lars Grunske

Received: date / Accepted: date

Abstract *Context:* Ensuring safety for any sophisticated system is getting more complex due to the rising number of features and functionalities. This calls for formal methods to entrust confidence in such systems. Nevertheless, using formal methods in industry is demanding because of their lack of usability and the difficulty of understanding verification results. *Objective:* We evaluate the acceptance of formal methods by Bosch automotive engineers, particularly whether the difficulty of understanding verification results can be reduced. *Method:* We perform two different exploratory studies. First, we conduct a user survey to explore challenges in identifying inconsistent specifications and using formal methods by Bosch automotive engineers. Second, we perform a one-group pretest-posttest experiment to collect impressions from Bosch engineers familiar with formal methods to evaluate whether understanding verification results is simplified by our counterexample explanation approach. *Results:* The results from the user survey indicate that identifying refinement inconsistencies, understanding formal notations, and interpreting verification results are challenging. Nevertheless, engineers are still interested in using formal methods in real-world development processes because it could reduce the manual effort for verification. Additionally, they also believe formal methods could make the system safer. Furthermore, the one-group pretest-posttest experiment results indicate that engineers are more comfortable un-

Arut Prakash Kaleeswaran
Cross-Domain Computing Solutions, Robert Bosch GmbH, Stuttgart, Germany, at the time
of the study: Corporate Sector Research, Robert Bosch GmbH, Renningen, Germany
E-mail: arutprakash.kaleeswaran@de.bosch.com

Arne Nordmann
Corporate Sector Research, Robert Bosch GmbH, Renningen, Germany
E-mail: arne.nordmann@neura-robotics.com

Thomas Vogel & Lars Grunske
Software Engineering Group, Humboldt-Universität zu Berlin, Berlin, Germany
E-mail: {thomas.vogel, grunske}@informatik.hu-berlin.de

derstanding the counterexample explanation than the raw model checker output. *Limitations:* The main limitation of this study is the generalizability beyond the target group of Bosch automotive engineers.

Keywords User study · Error comprehension · Counterexample interpretation · Formal methods · model checker

1 Introduction

The overarching goal of formal methods is to help engineers construct more reliable systems. – Clarke and Wing (1996)

Motivation. Research on formal methods has been continuing for more than three decades (Bowen and Breuer 2021). Initial applications of formal methods are introduced by Wing (1990) and Rushby (1993). In recent years, formal methods have been developed to analyze and verify complex safety-critical systems (Ferrari and ter Beek 2023; Kaleeswaran et al. 2022). Especially automated verification techniques based on formal methods are promising candidates (Baier and Katoen 2008; Grumberg and Veith 2008; Clarke et al. 2018a,b) to ensure the functional safety, for instance, of automotive systems. Even though the use of formal methods is considered to be a promising solution, industries are still hesitant to use it for real-world projects due to its complexity (Heitmeyer 1998; Abrial 2006; Bicarregui et al. 2009; Kossak et al. 2014; Jones and Thomas 2022; Ferrari and ter Beek 2023). To adopt formal methods in industry, usability and learnability are key factors (ter Beek et al. 2019; Reid et al. 2020). There exist several approaches that ease the use of formal methods. For example, property specification patterns (Dwyer et al. 1999; Konrad and Cheng 2005; Grunske 2008; Post and Hoenicke 2012; Autili et al. 2015) and structured natural language (Giannakopoulou et al. 2020) are convenient means to specify requirements to be translated into a temporal logic. Furthermore, the tools by Ratiu et al. (2021), Gerking et al. (2015), and Barbon et al. (2019) support performing verification in integration with a model checker. In a recent survey, we provide an overview of the state of the art in research on explaining counterexamples and how engineers are supported in interpreting counterexamples (Kaleeswaran et al. 2022).

With this work, we want to identify challenges and opinions on using formal methods in a concrete industrial setting, which is the identification of inconsistent specifications of safety-critical automotive systems at Bosch. With the rising number of features and functionalities, ensuring safety of such systems is a complex task that can be supported by formal methods such as model checking. However, adopting formal methods in industry is demanding because of lack of usability and the difficulty of understanding verification results obtained by model checkers. Therefore, we want to investigate whether a concrete approach to counterexample explanation eases the understanding of verification results, thus helping Bosch engineers in their daily work to specify and verify their specifications of safety-critical automotive systems.

Contributions. In this work, we present the results obtained from two different user studies with Bosch automotive engineers: (*Part 1*) *user survey* and (*Part 2*) *one-group pretest-posttest experiment*. 41 participants had taken part in the *user survey* and 13 participants in the *one-group pretest-posttest experiment*. From the *user survey*, we first collect challenges on identifying inconsistent specifications and concrete inconsistencies in a specification, and on maintaining refinement consistency between components. We further collect feedback and opinions in using formal methods. The results of the *user survey* are analyzed quantitatively and qualitatively.

Formal methods are not completely new to Bosch. They are used to define requirements as pattern-based specifications to support verification during product development (Post et al. 2012; Post and Hoenicke 2012). Additionally, we have presented a *counterexample explanation approach* that attempts to ease the use of formal methods by reducing the manual work and difficulty of interpreting the verification results generated by model checkers (Kaleeswaran et al. 2020). Thus, the motive of the *one-group pretest-posttest experiment* is to evaluate our *counterexample explanation approach* in an industrial setting at Bosch. Particularly, we investigate whether explaining a counterexample generated by a model checker in an understandable format would increase the use of formal methods among Bosch engineers.

We pre-registered our studies (Part 1 and 2) by a report submitted to and accepted by the Registered Reports Track at the International Conference on Software Maintenance and Evolution (ICSME) 2021¹ (Kaleeswaran et al. 2021). Both studies have been conducted following the same research protocol as described in the registered report (Kaleeswaran et al. 2021) without any modification to the design of the studies.

Outline. We introduce the background and terminology in Section 2, and discuss related user studies concerning formal methods in Section 3. We provide an overview of the counterexample explanation approach in Section 4. In Section 5, we outline the research questions, the design, execution plan, target participants, and analysis plan of the studies. We present the results of the *user survey* and *one-group pretest-posttest experiment* in Sections 6 and 7, and discuss and interpret them in Section 8. Threats to validity are discussed in Section 9. Finally, we conclude and outline the future work in Section 10.

Summary of results. From the results of the user survey, we found out that understanding formal notations (Section 6.2), identifying inconsistent specifications and understanding inconsistencies (Section 6.3), as well as maintaining the consistency of refined specifications and verifying the refinement consistency (Section 6.4) to be difficult for engineers. Further, the majority of participants answered positively that formal verification could support safety analysis and make a system safer (Section 6.5), formal methods are potential

¹ <https://icsme2021.github.io/cfp/RegisteredReportsTrack.html>

candidates to use in the real-world development process, and usage of formal methods could be increased by improving understanding of formal notations (Section 6.6).

From the results of the one-group pretest-posttest experiment, we found that participants obtain a good understanding of the inconsistencies with our proposed counterexample explanation (Sections 7.4 and 7.5) Further, analyzing the feedback from the participants, the majority of them find that the counterexample explanation provides a better and quicker understanding than the counterexample generated by the model checker (Sections 7.6–7.8).

2 Background and Terminology

In this section, we introduce the background and terminology of formal methods, model checking, contract-based design, which are relevant for our studies.

Formal methods. By formal methods we refer to models, *e.g.*, SysML (Friedenthal et al. 2014), and Kripke structures used as input to verification tools (McMillan 1999), formal specifications of requirements, *e.g.*, expressed in natural language-like statements using property specification patterns (Dwyer et al. 1999), or directly in a temporal logic such as Linear Temporal Logic (LTL) (Pnueli 1977), and to automated tools to perform the verification. Examples of such tools are model checkers, *e.g.*, NuSMV (Cimatti et al. 2000), theorem provers, *e.g.*, Isabelle (Paulson 1994), and solvers, *e.g.*, Z3 (de Moura and Bjørner 2008).

Model checking. Considering the verification as a model checking problem, a specification is expressed in a temporal logic (ϕ) and a system is modeled as a Kripke structure (K). Both are the input for a model checker. The model checker verifies whether the given system model (K) satisfies the given *property or specification* (ϕ), that is, $K \models \phi$ (Baier and Katoen 2008; Clarke et al. 2018a). If ϕ is not satisfied by K , the model checker generates a *counterexample* describing an execution path in K that leads from the initial system state to a state that violates ϕ , where each state consists of system variables with their values. Based on the counterexample, a user can manually localize the fault in K that causes the violation of ϕ .

Contract-based design. Contract-based design (CBD) (Cimatti and Tonetta 2012; Kaiser et al. 2015) supports the automated verification of refinement consistency and correctness. In CBD, model checking is used to identify whether the top-level requirements of a system are consistently refined along the refinement of the system to components. Each component of a system is associated with a contract that precisely specifies the expected behavior of the component by assumptions, and the provided behavior by guarantees. If a component is refined to sub-components, its contract is also refined and assigned to its sub-components. Thus, all of the sub-components should satisfy the expected

behavior of the parent component. This corresponds to the correctness of the refined contracts and can be verified by model checking, which is known as the *refinement check* (Cimatti and Tonetta 2012).

3 Related Work

In the following, we discuss existing user surveys focusing on formal methods. Interviews with users of formal methods are conducted by Snook and Harrison (2001) to collect the impact on using formal methods on the company, products, and the development process. Furthermore, the interviews focus on various software engineering aspects such as scalability, understandability, and tool support. Rodrigues et al. (2018) perform a survey with 20 participants who use formal specifications to solve limitations of informal specifications. Khazeev et al. (2019) conduct a survey with the students of the Software Engineering program to examine the AutoProof tool and highlight the challenges associated with formal approaches.

There exists several surveys focusing mainly on particular application domains. Davis et al. (2013) conduct a survey with 31 participants in the aerospace domain. The survey collects barriers in using formal methods and also propose mitigation measures for the identified barriers. The studies by Ferrari et al. (2019) and ter Beek et al. (2019) focus on the railway domain. Ferrari et al. (2019) summarizes results that are suitable for system modelling and verification from surveying 114 primary studies, 44 participants from academia and industry, and eight projects. Similarly, ter Beek et al. (2019) collect the opinion of users on adopting formal methods in the same domain.

According to the recent user study by Gleirscher and Marmsober (2020) performed with 216 participants, the participants are inclined to use formal methods when sufficient training and tool support is available. They also present a systematic map of 35 existing studies that summarizes the opinions of the studies' authors on formal methods, as well as the motivation, research method, and results of the studies. Similarly, the study performed by Garavel et al. (2020) with 130 participants focus mainly on the use of formal methods in research, industry, and education in the past and present. Furthermore, the study summarizes the future direction of formal methods in industry, future target audience using formal methods, domains in which formal methods may have an impact, and competitive or alternative methods to formal methods.

In contrast to these studies, (1) we particularly focus on identifying challenges that engineers face in identifying inconsistent specifications rather than general challenges of using formal methods, and (2) we conduct the study with engineers who work on real-world projects in the automotive domain.

4 Counterexample Explanation in a Nutshell

Contract-based design (CBD) (Cimatti and Tonetta 2012) is a scalable solution to overcome a manual analysis by automatically and compositionally verifying

the consistency of system and requirement refinements using model checking. Thus, CBD provides assurances for consistent refinements early in the development process, which promises to ease corresponding testing activities at later stages. However, using such a formal method in industry is challenging due to usability issues, *e.g.*, the difficulty of understanding model checking results. Thus, we have proposed a counterexample explanation approach that eases the error comprehension of engineers—especially of non-experts in formal methods—if the refinement check fails. The approach generates a user-friendly explanation that localizes the fault at the levels of requirements and components. Examples of a CBD and explanations of counterexamples will be given in the context of the study in Section 7.

The counterexample explanation approach comprises of six steps illustrated in Figure 1. Steps ① and ⑥ are performed manually while the other steps are completely automated. Step ① is the translation of CBD by importing SysML models from Rhapsody and DNG requirements into FASTEN (Ratiu et al. 2021). FASTEN is an open-source platform to experiment with rigorous modeling of safety-critical systems. Translating the (largely informal) requirements from DNG into contracts is a manual effort, further detailed in the context of Bosch by Post et al. (2012); Post and Hoenicke (2012). CBD languages provided by FASTEN allow us to model component-based architectures, requirements as contracts (assumptions and guarantees), and refinements, hence creating a CBD. After the design, in Step ②, FASTEN automatically translates a CBD to a formal system model K and refinement specification ϕ that allows model checking by NuSMV/nuXMV in Step ③. This refinement specification follows the scheme by Cimatti and Tonetta (2012) who define a refinement check by a set of LTL formulae (Kaleeswaran et al. 2020, Section 2). If the model checker identifies any refinement inconsistency during the verification, it returns the violated LTL refinement specification and the counterexample.

Taking the violated LTL refinement specification and counterexample as input, the counterexample explanation approach extracts erroneous parts in Step ④. To extract such parts, we identify (i) the *inconsistent specifications* in the violated LTL refinement specification, (ii) *inconsistent sub-specifications* in the inconsistent specifications, (iii) *erroneous contracts and components* by using the inconsistent specifications and by referring to the refinement formula, (iv) *erroneous states* in the counterexample, and (v) *erroneous variables* by using the inconsistent sub-specifications (cf. ④A – ④E in Figure 1).

Finally in Step ⑤, a statement is generated explaining inconsistency along with the counterexample highlighting *erroneous states* and *erroneous variables*. The generated statement consists of the *erroneous components*, violated contract information, and *inconsistent specifications* expressed in a pattern-based language referring to requirements and SysML model being the initial user input of the approach (cf. ① in Figure 1). Finally, the *inconsistent sub-specifications* are highlighted in the pattern-based expression. With the statement, the engineer gets a high-level understanding of the refinement inconsistency. Further, the counterexample with highlighted erroneous parts supports the engineer in understanding the erroneous behavior of the system. Finally,

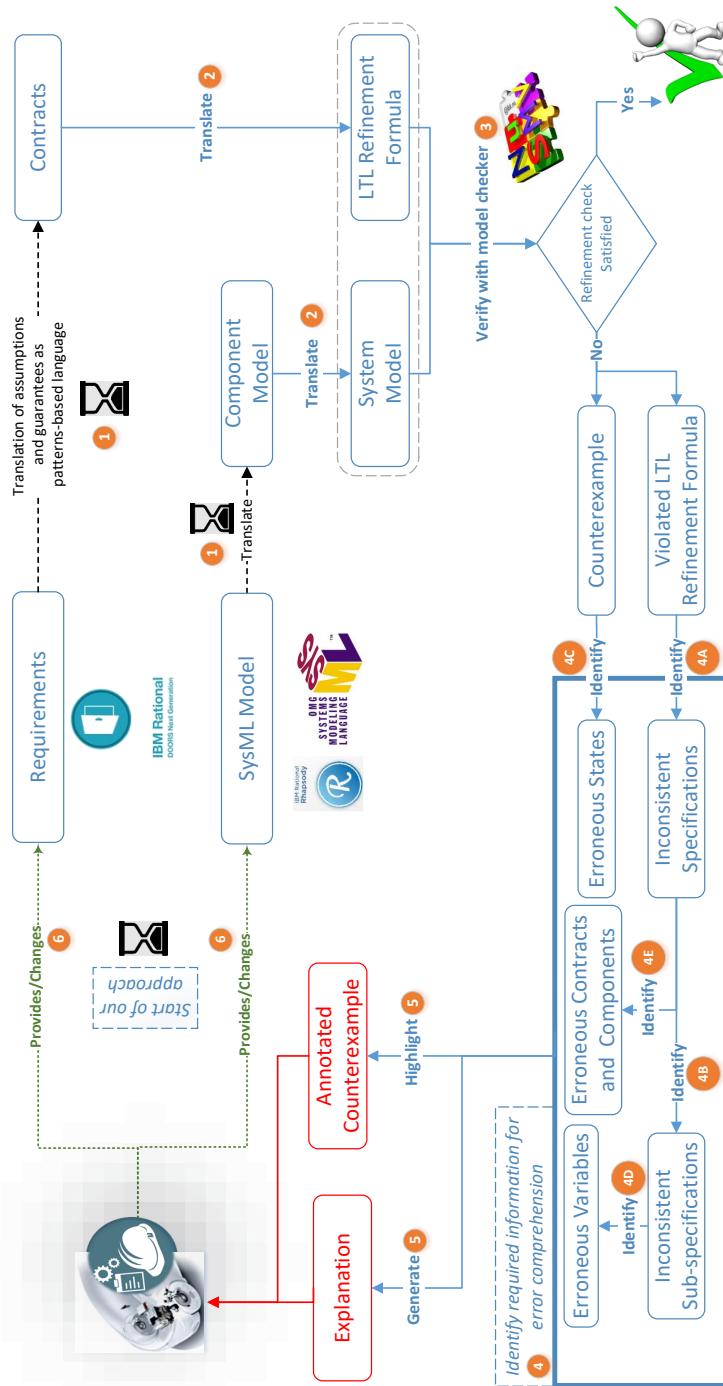


Figure 1: Overview of the counterexample explanation approach.

the engineer can correct the refinement inconsistencies by remodeling the component model and changing the requirements (cf. ⑥ in Figure 1). To ensure the correctness of the changes, the engineer re-verifies the changed refinement.

5 Research Method

In this section, we discuss the research questions, design, execution plan, target participants, and analysis plan of the studies.

5.1 Research Questions

Our study aims to explore and understand the challenges in identifying inconsistent specifications, and the acceptance of formal methods by Bosch automotive engineers. Therefore, this user study has two significant goals: (**G1**) to understand challenges faced by Bosch engineers when identifying inconsistent specifications, and challenges along with their opinions to use formal verification or formal methods in real-world development processes, and (**G2**) to explore whether Bosch engineers are interested in using formal methods, particularly model checking, in real-world development processes if the difficulty of understanding model checking results is reduced by our counterexample explanation approach. Considering these two goals, we formulate the following three research questions:

RQ1: *To what extent do engineers face challenges in identifying inconsistent specifications in formal models that are introduced during the refinement of a system?*

With this RQ we want to investigate whether:

- (I1) Understanding formal notations is difficult for engineers.
- (I2) Identifying inconsistent specifications that are introduced during a refinement of a top-level specification is difficult.

RQ2: *To what extent the identification of inconsistent specifications and usage of formal methods prove beneficial to a real-world development process?*

With this RQ we want to investigate whether:

- (I4) Usage of formal verification or formal methods is beneficial in a real-world development process.
- (I5) Identifying inconsistent specifications is beneficial in a real-world development process.

RQ3: *To what extent do engineers prefer to use formal methods (model checkers particularly) if the difficulty is reduced for understanding verification results to identify inconsistent specifications?*

With this RQ we want to investigate whether:

- (I5) The counterexample explanation approach eases the comprehension when compared to interpretation of the raw model checker output for engineers with a formal methods background.
- (I6) It is possible for engineers with a background in formal methods to identify and fix inconsistent specifications based on the counterexample explanation approach.
- (I7) The counterexample explanation approach can promote formal verification and usage of model checking in real-world development processes.

5.2 Variables

To attain the goals **G1** and **G2**, we perform two different types of exploratory user studies as shown in Figure 2. The first study is the *user survey (Part 1)*, and the second study is a *one-group pretest-posttest experiment (Part 2)*.

5.2.1 Variables of Part 1: User Survey

Our user survey evaluates the research questions **RQ1** and **RQ2**. The independent variables of *Part 1* are *participants' professional background and experience*. The dependent variables are different for each research questions. For **RQ1**, the dependent variable is the *difficulty in understanding* that infers understanding formal notations and identifying inconsistent specifications by engineers are difficult. Similarly, the dependent variable for **RQ2** is the *increase in confidence in system safety*, that is, the identification of inconsistent specifications and use of formal methods in real-world development processes can make systems safer.

5.2.2 Variables of Part 2: One-Group Pretest-Posttest Design

According to Babbie (2016), an experimental stimulus (also called an intervention) is the independent variable. In the one-group pretest-posttest design, we use our counterexample explanation approach as an intervention. Therefore, it serves as the independent variable of *Part 2*. Further, we evaluate **RQ3** based on the following four attributes that serve as dependent variables for *Part 2*:

1. *Better understanding*: Does the counterexample explanation approach allow engineers to understand model checking results and identify inconsistencies more effectively?
2. *Quicker understanding*: Does the counterexample explanation approach allow engineers to understand model checking results and identify inconsistencies more efficiently?
3. *Confidence*: Does the counterexample explanation approach make engineers more confident in their understanding of the system and its inconsistency respective to safety?

4. *No value*: This attribute is inversely related to the above attributes. Will the counterexample explanation approach provide no or only minimal value to real-world projects?

For each of the four attributes, the participants are asked one question in the pretest and posttest in order to collect their opinions about the attributes. Table 3 lists the questions asked to the participants and the scale of possible answers in both tests. Particularly, the questions PRQ1 in the pretest and POQ1 in posttest target the attribute *Better understanding*, PRQ2 and POQ2 target *Quicker understanding*, PRQ3 and POQ3 target *Confidence*, and PRQ4 and POQ4 target *No value*. Finally, comparing the results gathered from both the pretest and posttest, we can investigate the participants' opinions on understanding inconsistencies using the proposed counterexample explanation and the raw counterexample generated by the model checker.

Table 1: Questionnaire of our user survey (Part 1). A scale is either nominal (N) or ordinal (O). Labels are either not applicable (NA) or they refer to one of the scales (LS) defined in Table 2.

#	Questions	Scale	Label
Demographic Questions			
Q1	Designation/Role/Assignment	N	NA
Q2	Rate your knowledge of formal methods	O	LS1
Q3	How many years have you used formal methods in your daily work?	O	LS2
Q4	Indicate the applications/products/projects you worked on using formal methods	N	NA
Q5	How many years have you worked in the safety domain?	N	LS2
Q6	Indicate the applications/products/projects you worked on focusing on safety aspects	O	NA
Main Survey Questions			
Q7	How easy is it for you to understand formal notations?	N & O	LS3
Q8	What is your opinion on using formal verification?	N	NA
Q9	How easy is it for you to identify inconsistent formal specifications (where the consistency is)?	N & O	LS3
Q10	How easy is it for you to identify the inconsistencies in formal specifications (the cause for the inconsistency)?	N & O	LS3
Q11	How fast could you identify inconsistent formal specifications?	N & O	LS4
Q12	What are the challenges that you face in order to identify inconsistent formal specifications?	N	NA
Q13	What sort of methods have you used to identify inconsistent formal specifications?	N	NA
Q14	How easy is it for you to maintain consistency when refining formal requirements for sub-components of a system architecture?	N & O	LS3
Q15	How hard is it for you to check the consistency of formal requirements that are associated with components of a system architecture?	N & O	LS3
Q16	In your opinion, will the usage of formal verification make systems safer?	N & O	LS5
Q17	In your opinion, can formal verification be an add-on to the functional safety methods to ensure safety?	N & O	LS5
Q18	In your opinion, how beneficial is the identification of inconsistent formal specifications for a safety analysis?	N & O	LS5
Q19	Can you imagine using formal methods if understanding of formal notations is made easier?	N & O	LS5
Q20	Do you think formal methods are usable in real-world development processes?	N & O	LS5

Table 2: Scales used for our user study.

Label	Type	Answer Scales
LS1	Expertise	Novice, Advanced Beginner, Competent, Proficient, Expert, Mastery, Practical Wisdom, No Opinion
LS2	Experience	< 1, 1 to < 2, 2 to < 4, 4 to < 6, 6 to < 8, 8 to < 10, < 10, No Experience
LS3	Agreement	Extremely Hard, Hard, Slightly Hard, Neither Hard nor Easy, Slightly Easy, Easy, Extremely Easy, No Opinion
LS4	Agreement	Extremely Fast, Fast, Slightly Fast, Neither Fast nor Slow, Slightly Slow, Slow, Extremely Slow, No Opinion
LS5	Likelihood	Definitely, Very Probably, Probably, Neither Probably nor Possibly, Possibly, Probably Not, Definitely Not, No Opinion
LS6	Agreement	Strongly Agree, Agree, Somewhat Agree, Neither Agree nor Disagree, Somewhat Disagree, Disagree, Strongly Disagree, No Opinion
LS7	Usefulness	Exceptional, Excellent, Very Good, Good, Fair, Poor, Very Poor, No Opinion

5.3 Design of the User Study

In this section, we describe the design, questionnaires, and tools used for the *user survey (Part 1)* and the *one-group pretest-posttest experiment (Part 2)*.

5.3.1 Part 1: User Survey

For *Part 1*, we use a cross sectional survey (Kitchenham and Pfleeger 2008) to collect data from engineers to achieve goal **G1**. For planning and conducting this user survey, we follow the guidelines by Neuman (2014) (majorly Chapter 7), Kitchenham and Pfleeger (2008), and Fink (2003). In addition, we follow the guidelines by Robson and McCartan (2016) (Chapter 11), and Babbie (2016) (Chapter 9) for the questionnaire construction. Furthermore, we also refer to and adapt some of the questionnaires from existing user surveys (Gleirscher and Marmsoler 2020; Garavel et al. 2020). The questionnaire of our user survey is shown in Table 1. Responses to each question are either collected as qualitative statements, and/or they follow predefined eight-point Likert scales shown in Table 2.

5.3.2 Part 2: One-Group Pretest-Posttest Experiment

Part 2 of our study is an exploratory pre-experimental study following a *one-group pretest-posttest experiment* design to attain goal **G2**. We follow the guidelines by Campbell and Stanley (1963) to conduct this part of our study.

One of the main drawbacks of using a one-group pretest-posttest design is that it does not meet the scientific standards of an experimental design. The pre-experimental study designs does not have a control group like a true experiment (Wohlin et al. 2012). Thus, comparison and generalization of the results based on the provided intervention/stimulus may not be possible. However, we intend to use this pre-experimental study design because of the scarcity of participants. To find a considerable number of participants (30 to 40) with knowledge of formal methods and model checkers inside an industrial organization is way too ambitious. Performing a true experiment with a lower number

Table 3: Questionnaire of the one-group pretest-posttest study (Part 2). A scale is either nominal (N) or ordinal (O). Labels are either not applicable (NA) or they refer to one of the scales (LS) defined in Table 2.

#	Questions	Scale	Label
Demographic Questions			
DQ1	Rate your knowledge of formal methods	O	LS1
DQ2	How many years have you used formal methods in your daily work?	O	LS2
Task Questions			
TQ1	Do you think this use case is difficult?	N & O	LS5
TQ2	How difficult was this use case for you to understand?	N & O	LS5
TQ3	Do you think you have understood results from the model checker? (This question is only for pretest)	N & O	LS5
TQ4	Do you think you have understood the explanations? (This question is only for the posttest)	N & O	LS5
TQ5	Of the following list, please select the inconsistent components.	N	NA
TQ6	Of the following list, please select the inconsistent specifications.	N	NA
TQ7	Please explain the reason that makes the specifications inconsistent from your understanding.	N	NA
TQ8	Please provide a solution to fix the inconsistency from your understanding.	N	NA
TQ9	Please provide a nominal behavior that is expected in the counterexample's erroneous states from your understanding.	N	NA
Understanding Model Checker Outputs (Pretest)			
PRQ1	The results from the model checker allow me to understand the inconsistencies.	N & O	LS6
PRQ2	Such a result from the model checker could save me time.	N & O	LS6
PRQ3	The results from the model checker make me confident that I really understand the inconsistencies that I am investigating.	N & O	LS6
PRQ4	The value added by such a result from the model checker will be minimal.	N & O	LS6
Understanding Counterexample Explanation (Posttest)			
POQ1	An approach like counterexample explanation allows me to better understand inconsistencies.	N & O	LS6
POQ2	An approach like counterexample explanation saves me time.	N & O	LS6
POQ3	An approach like counterexample explanation makes me more confident that I really understand the inconsistencies that I am investigating.	N & O	LS6
POQ4	The value added by an approach like counterexample explanation is minimal.	N & O	LS6
Counterexample Explanation Features (Ratings)			
FQ1	Translation of specifications from formal temporal format to natural language-like format.	N & O	LS7
FQ2	Listing inconsistent specification.	N & O	LS7
FQ3	Highlighting sub-parts of the inconsistent specifications that leads to an inconsistency.	N & O	LS7
FQ4	Providing the component name that belongs to the inconsistent specifications.	N & O	LS7
FQ5	Providing an expected nominal behavior in the explanation for the corresponding erroneous states and variables of the counterexample.	N & O	LS7
FQ6	Highlighting the erroneous states and variables in the counterexample.	N & O	LS7
Feedback (After completion of the experiment)			
FE1	Are inconsistencies easier to understand with the results created by the counterexample explanation approach in comparison those of the original model checker?	N & O	LS5
FE2	What challenges did you face while analyzing inconsistencies in a specification with the proposed approach?	Nominal	NA
FE3	Do you think it is easy to maintain consistency with the proposed counterexample explanation while refining requirements into requirements for sub-components?	N & O	LS3
FE4	Do you think the proposed counterexample explanation approach is usable in real-world development processes?	N & O	LS5
FE5	Would you consider using formal methods with our approach in real-world projects?	N & O	LS5
FE6	Would you consider using the presented approach in your project? If so, please name the project and a contact person?	N & O	LS5
FE7	Do you think presenting a list of possible suggestions/fixes would be helpful to understand and fix inconsistencies?	N & O	LS5
FE8	Suggestions for further improvements.	Nominal	NA

of participants raises the threat to external validity. Therefore, we intend to perform a one-group pretest-posttest experiment with Bosch automotive engineers that allows us to capture results from real-world user behavior, even with a limited number of participants. However, the pre-experimental study has several internal and external threats to be considered. In Section 9, we discuss corresponding threats raised by (Campbell and Stanley 1963, Table 1).

Along with the guidelines by Campbell and Stanley (1963), we refer to the protocol by Zaidman et al. (2013) for a one-group pretest-posttest experiment. They evaluate a tool called *FireDetective* that supports understanding of Ajax applications at both the client-side (browser) and server-side. Their evaluation is performed using two user study variants (i) pretest-posttest user study, and (ii) a field user study, where the former is performed with eight participants and the latter is performed with two participants. In our study, we perform the one-group pretest-posttest experiment with Bosch automotive engineers and discard the field user study for our evaluation. The questionnaire shown in Table 3 is used for the one-group pretest-posttest study (*Part 2* of our overall study). Similar to *Part 1*, responses to each question are either qualitative statements, selections of options on predefined eight-point Likert scales (Table 2), or a combination of both.

5.4 Tools Used for the Study

Both studies are performed remotely due to the COVID-19 pandemic. In such a setting, it would not be easy to capture the time that participants spent on the study. For example, there could be a situation where participants could have taken a break or could respond to some urgent emails while during the study. Thus, we asked separately a question for the time taken by the participants to conduct the pretest and posttest. To perform the studies and collect the answers by the participants, we use *Microsoft Forms for Excel* that is easily accessible within the company and already familiar to the participants. The results are stored in a Microsoft Excel file, which we use to perform the analysis. All content-wise explanations for this study are provided as a video that are accessible Bosch internally via an online platform called *BoschTube*.

5.5 Participants

Our counterexample explanation approach focuses on enhancing safety analysis for automotive systems (Kaleeswaran et al. 2020). Thus, we are interested in performing this user study only with automotive engineers have at least basic knowledge of formal methods, particularly engineers working on system development, requirement elicitation, and safety analysis.

The target population for our study is very specific and thus, it is hard to make a finite list of participants by applying probabilistic sampling. According to Kitchenham and Pfleeger (2008), when a target population is very specific

and limited, non-probabilistic sampling can be used to identify the participants. Therefore, we intend to use two non-probabilistic sampling methods for *Part 1* of our study, namely, *convenience sampling* and *snowball sampling*. Further, we invite participants with knowledge on formal methods for *Part 2* of our study by filtering the participants of *Part 1* based on the responses to the demographic questions Q1 to Q3 listed in Table 1.

First, we start with the convenience sampling for *Part 1*. We send e-mails with the survey link to participants collected through department mailing lists and community mailing lists of all relevant Bosch business units. We perform snowball sampling with the accepted participants by asking for further potential participants at the end of the survey. In the e-mail invitation, we explicitly mention that the anonymity of results will be preserved. So, while summarizing analysis results, we remove all personal, product- and project-related information. For both the studies, the reminder mail is sent three times in the interval of one week.

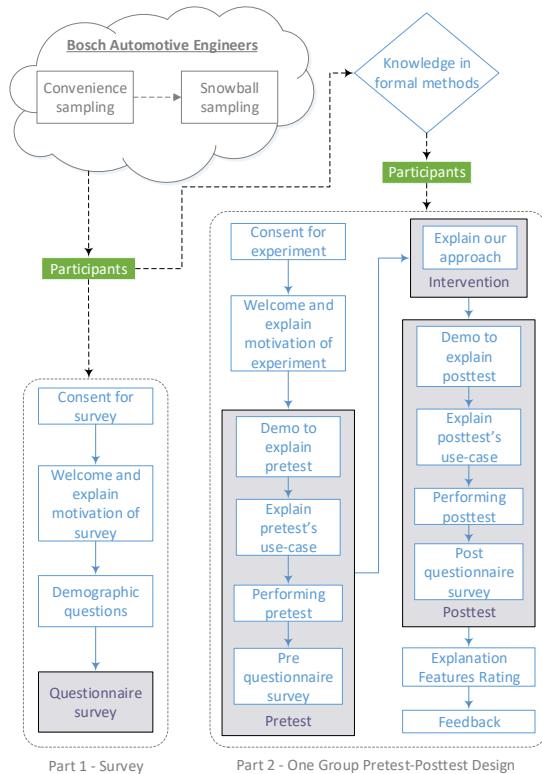


Figure 2: Overview of the study. *Part 1* is a user survey performed with a wide range of participants. *Part 2* is a one-group pretest-posttest experiment performed with engineers having knowledge in formal methods. Gray color boxes indicate the main tasks, *i.e.*, questionnaire survey, pretest and posttest.

5.6 Execution Plan

In this section, we describe the execution plan of the *user survey (Part 1)* and the *one-group pretest-posttest experiment (Part 2)* depicted in Figure 2.

5.6.1 Execution Plan of Part 1

The *user survey (Part 1)* comprises four steps (Figure 2). First, we notify participants regarding the data processing agreement. Additionally, we also state explicitly that their names, project- and product-related information will be removed while results are shared for evaluation. Then we show a video, welcoming the participant and explaining the background and motivation of this survey. Then, we ask participant to answer the demographic questions (Q1 to Q6 in Table 1), and further the main survey questions (Q7 to Q20 in Table 1). Finally we conclude the survey with a thanks note.

5.6.2 Execution Plan of Part 2

For the one-group pretest-posttest experiment, we invite participants from *Part 1* who have knowledge in formal methods. Similar to *Part 1*, *Part 2* starts with a data processing agreement, followed by a background and motivation video. Our one-group pretest-posttest experiment is executed with the invited participants as follows: a pretest experiment, then intervention, and finally the posttest experiment.

Pretest. The pretest experiment starts with a video demonstrating the experiment with a simple example of an OR-gate and the behavior of the OR-gate. After that, another video introduces an airbag system with the corresponding system model and specification, that serves as a use case for the actual pretest experiment. During the actual experiment, the participant analyzes the violated specification and the counterexample returned by the model checker to understand the inconsistent parts of the specification. Furthermore, based on the understanding, the participant answers the task questions (TQ1 to TQ9 except of TQ4 in Table 3). Finally, the pretest is concluded by answering the pre-questionnaire survey questions PRQ1 to PRQ4 listed in Table 3.

Intervention. After the pretest experiment, a video introducing the counterexample explanation approach Kaleeswaran et al. (2020) is shown to the participants. This serves as an intervention in our study.

Posttest. Like the steps followed for the pretest experiment, the posttest experiment starts with a demonstration video with the same use case of the OR-gate, but this time with the counterexample explanation approach. This is followed by a video that introduces the electronic power steering system (EPS), a commercial Bosch product, with the corresponding system model and specification. Then the participants interpret the explanation provided by the

counterexample explanation approach to understand the inconsistency. Based on the explanation, participants answer the task questions (TQ1 to TQ9 except of TQ3 in Table 3). Subsequently, they answer the post-questionnaire survey questions POQ1 to POQ4 listed in Table 3. After completing the posttest experiment, participants rate the features (FQ1 to FQ6 in Table 3) provided by the counterexample explanation approach and respond to the feedback questions (FE1 to FE8 in Table 3). Finally, *Part 2* of our study concludes with a thanks note to the participants.

5.7 Presentation of the Analysis Results

To obtain the results from the study, we follow the recommendation by Robbins and Heiberger (2011). We use normal, grouped and stacked bar charts to plot the results. Qualitative statements received from participants are gathered, organized, and summarized individually for every question. We summarize the qualitative statements through the following three steps: *(i) Microanalysis*: The first author goes through the individual answers from the participants and assigns labels to the statements. The rest of the authors validate the initial labels and provide feedback for improvement. At the end of this step, all authors come to a mutual agreement on the initial labels. *(ii) Categorization*: Based on the feedback for improvement, the first author performs second iteration. As a result, a set of themes are extracted which are deemed to be essential. *(iii) Saturation*: This is the final step where all the authors come to the final agreement on labels, themes, and summarized statements. Since the qualitative statement is a medium to express an individual opinion, the categorization of labels are associated with the demographic answers. For example: “*an engineer who has seven years of experience states that the counterexample explanation approach can promote the usage of model checkers among system engineers*”.

6 User Survey (Part 1): Results and Analysis

In this section, we present and analyze the results of the user survey (*Part 1*). We gathered answers to the questionnaire shown in Table 1 from 41 participants.

6.1 Participants

We first present demographic information of the participants that we obtained from the first six questions of the questionnaire (*Q1* to *Q6* in Table 1).

6.1.1 Experience in Formal Methods and Safety

The experience of the participants in using formal methods and their experience focusing on safety are collected through the questions *Q3* and *Q4* (Ta-

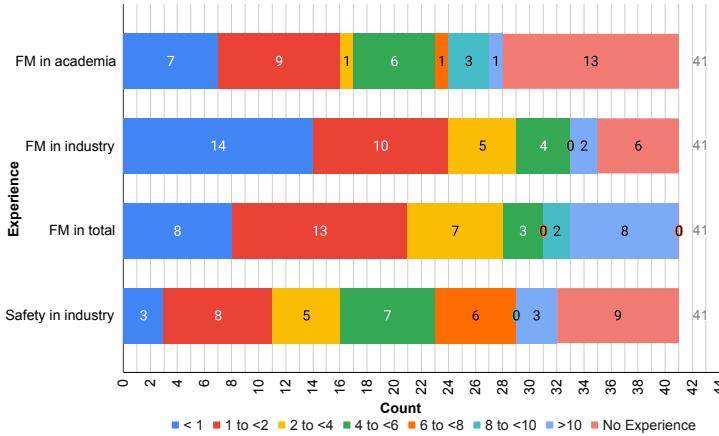


Figure 3: Experience of participants in formal methods gained in academia, industry, and the overall experience, along with industrial experience on safety.

ble 1), which are answered in scale *LS2* (Table 2). Participants are asked to fill in their experience in formal methods gained individually in academia, and in industry, and their overall experience in academia and industry combined together with *Q3*, while *Q4* collects industrial experience in safety. Figure 3 represents the responses for experience in formal methods and safety.

From Figure 3, it is clear that all the 41 participants have gained knowledge in formal methods in academia or industry *to some extent*. When excluding 13 participants who have no academic experience in formal methods, the predominant number of participants (82% of all participants who have academic experience) have experience of <6 years. Likewise, excluding 6 participants who do not have industrial experience in formal methods, the number of participants who have experience of <2 years is dominant (40% of all participants who have industrial experience). 29% of all participants with some experience in industry only have 1 to <2 years of experience.

In summary, 52% of all participants have <2 years, 24% have 2 to <6 years, no participants with 6 to <8 years, and 24% have >8 years of experience in formal methods. Looking at the participants' experience on safety, the results are scattered, with no clear majority. Excluding the 9 participants who do not have any experience in safety, 41% of all participants are experienced 1 to <4 years, further 41% are experience 4 to <8 years, 9% of participants have <1 year of experience, and the final 9% have >10 years of experience.

6.1.2 Knowledge of Formal Methods

Question *Q1* (Table 1) asks participants to rate their knowledge in formal methods according to the scale *LS1* (Table 2). Figure 4 shows the results. All participants have rated their knowledge within the scale *novice* to *expert*, while no participant provided a rating of *mastery* and *practical wisdom*. The major-

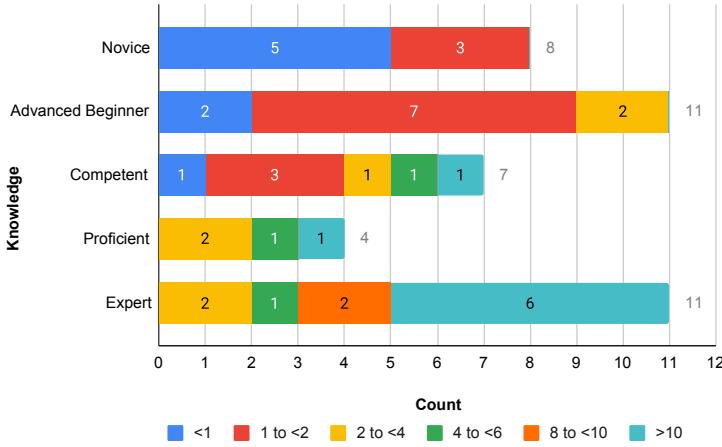


Figure 4: Knowledge of the participants categorized based on their overall experience in formal methods.

ity of participants rate their knowledge in formal methods as *advanced beginner* and *expert* with 27% of all participants each. Further, eight participants (20%) rate themselves as a *novice*, seven participants (17%) as *competent*, and four participants rate themselves to be *proficient*.

Figure 4 also presents participants' knowledge together with their years of experience in formal methods. The majority of participants with an experience of <1 year rated themselves as *novice*, with 1 to <2 years of experience as *advanced beginner*, and those with 8 to <10 years and >10 years of experience rated themselves as an *expert*. However, the participants who have experience of 2 to <4 years and 4 to <6 years have distributed their rating among various classes. Participants with experience of 2 to <4 years rated themselves from *advanced beginner* to *expert*, participants with 4 to <6 years of experience rated themselves from *competent* to *expert*.

6.1.3 Designation

Designations of the participants are collected with a free text field for question *Q2* (Table 1), depicted in Figure 5a. The majority of the participants (11 participants, 27%) consider themselves as *safety manager/engineer*, 22% as *systems engineer*, and 20% as a *research engineer*. Further, 11 participants (27%) are either an expert or an architect in either safety, system, software, or verification.

6.1.4 Industrial Application

Questions *Q4* and *Q6* in Table 1 assess the industrial applications, to which participants applied formal methods or worked on safety aspects. Identified clusters of applications are shown in Figure 5b. Formal methods are used

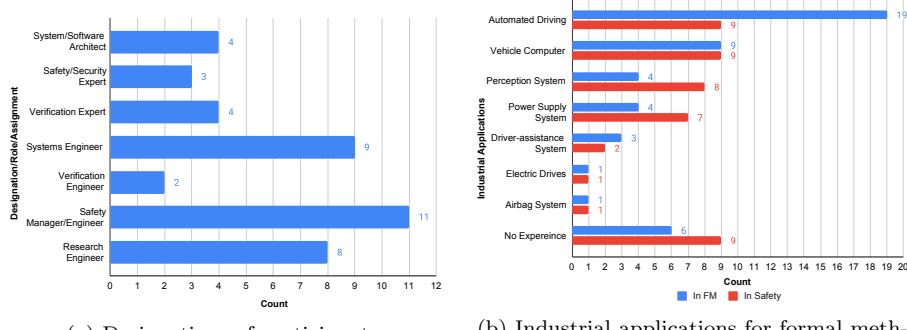


Figure 5: Designations and industrial applications of participants.

mostly for *automated driving* applications (19 participants) and *vehicle computer* (9 participants). These are also the most-called applications for safety aspects (9 participants each), followed by *power supply system* (8 participants) and *driver-assistance system* (7 participants).

6.2 Understanding Formal Notations

With question *Q7* (Table 1), we collect the participants' opinions on understanding formal notations. The possible answers follow the scale *LS3* (Table 2) and further allow comments as free text. Table 4 shows the results according to the scale. The majority (80% of all participants) is almost equally distributed between answers *hard* and *slightly easy*. 44% find formal notations between *slightly hard* and *extremely hard*, 34% find formal notations between *slightly easy* and *extremely easy*. In the received comments, the majority of participants agree that understanding formal notations gets easier with more usage and experience, and that it is highly dependent on the focus of the system domain such as automotive and railway. A supporting statement from a participant answering *slightly easy* states that “*If I am familiar with the formal language in which the formal notations are written (e.g., first-order language), typically it is easy.*”

Figure 6 shows responses for understanding formal notations based on the participants' knowledge in formal methods (*cf.* Section 6.1.2) and designation (*cf.* Section 6.1.3). For the following discussion, we cluster answers among *extremely hard*, *hard*, and *slightly hard* as “harder” and the ones among *extremely easy*, *easy*, and *slightly easy* as “easier”.

Table 4: Understanding formal notations.

Agreement	Extremely Hard	Hard	Slightly Hard	Neither Hard nor Easy	Slightly Easy	Easy	Extremely Easy	No Opinion
Count	2 (4%)	8 (20%)	8 (20%)	9 (22%)	8 (20%)	5 (12%)	1 (2%)	0

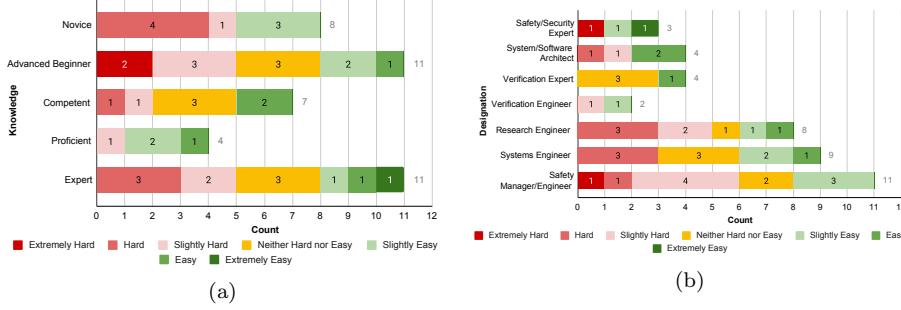


Figure 6: Results for understanding formal notations grouped by (a) the knowledge of participants in formal methods and (b) the designation of participants.

Understanding of formal notations based on knowledge in formal methods. Looking at *novices* and *advanced beginners* in Figure 6a, the predominant number of participants perceive understanding of formal notations as *harder*. Among the 11 *experts*, five participants perceive understanding formal notations to be *harder*, while three participants perceive understanding formal notation as *easier*. This is notable because even the majority of *experts* perceive understanding of formal notation to be *harder*. A participant rated as being an *expert* and answered *hard* mentions understanding notations used in logics like LTL, CTL, or TCTL are pretty straightforward, but using them to formalize real-world requirements is harder. Another *expert* who answers *extremely easy* states that using approaches like pattern-based languages (Dwyer et al. 1999) helps to ease understanding formal notations.

Understanding of formal notations based on participants' designation. Participants with designations of a *system/software architect*, *systems engineer*, and *verification engineer* perceive formal notations to be *harder* to understand while the same number of participants from each of the three mentioned categories vote for formal notations to be *easier* for understanding. A verification expert highlights: “*In general, formal notations are clear and precise. However, even for very experienced engineers, some formal notations (for example in temporal logics) are hard to understand on the semantic level, i.e., really telling what the formula means for the system at hand.*” Considering the designations of a *safety/security expert* and *verification expert*, the majority of the answers (excluding *neither hard nor easy*) opt for understanding formal notations to be *easier* with two out of three participants and one participant in each designation. For the remaining designations of a *safety manager/engineer* and *research engineer*, the majority of participants answer that understanding notations are *harder* with six out of 11 participants and five out of eight participants, respectively. A *systems engineer* state that “*I have never really learned formal notation; thus, I learned it on the job. A real introduction might have turned out helpful.*”

Summary. 44% of all participants answer understanding formal notations to be hard, only 34% of participants perceive understanding of formal notations as easy. From the answers received as free text, it is clear that experience plays a major role in understanding formal notations.

6.3 Inconsistent Formal Specifications

In this section, we discuss difficulties in identifying inconsistent specifications, understanding inconsistencies, the time taken to identify inconsistent specifications, as well as challenges and different methods to identify inconsistent specifications based on the answers to questions *Q9* to *Q13* (Table 1).

6.3.1 Identifying Inconsistent Formal Specification

With question *Q9*, we assess difficulties of identifying inconsistent formal specifications. Answers are given according to the scale *LS3* (Table 2) and as free text. The results of the answering 41 participants are shown in Table 5. Notably, a majority (51%) perceive the identification of inconsistent formal specifications as *hard*, with a total of 73% perceiving it as at least *slightly hard*.

From the free-text responses, 13 participants state that the effort of identifying inconsistent specifications highly depends on the kind of specification, as well as the complexity of the specification and system. For example, a *verification expert* who answers *neither hard nor easy* states that identifying inconsistent specification “depends on the size and kind of specification, e.g., debugging temporal logic is not trivial.” Regarding complexity of the specification, a *systems engineer* highlights an example that it “depends on the complexity of the formal specification itself and how well the inconsistency is hidden, e.g., $x < 10$ and $x > 20$ is easy to spot, but if you replace one x with y and link them somewhere else, it’s already hard to find.” However, on contrary, an interesting point is highlighted by a *verification expert*, stating that one “can’t really say, [as it] highly depends on the complexity of the application. With respect to model checking activation/deactivation of autonomous driving functions we realized that even small state machines can contain possibly critical errors. Also, these errors would have never been identified by classic testing (test runs, simulation).”

Identification of inconsistent formal specifications based on participants knowledge in formal methods. Figure 7 depicts the responses gathered for identifying inconsistent specifications together with the participants’ knowledge in formal methods. The count of responses as *harder* is dominating all the classes

Table 5: Results for the difficulty of identifying inconsistent specifications.

Agreement	Extremely Hard	Hard	Slightly Hard	Neither Hard nor Easy	Slightly Easy	Easy	Extremely Easy	No Opinion
Count	4 (10%)	21 (51%)	5 (12%)	7 (17%)	0	3 (7%)	1 (3%)	0

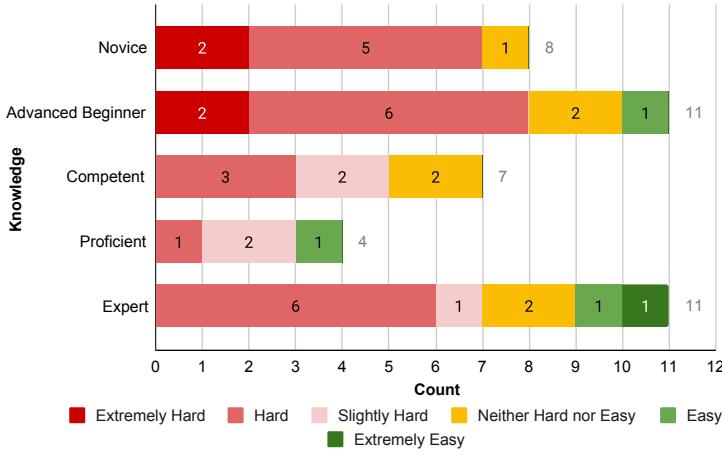


Figure 7: Results for identifying inconsistent formal specifications grouped by the participants' knowledge in formal methods.

ranging from *novice* to *expert*. Among all participants who answer identifying inconsistent specifications is *easier*, the majority is either *proficient* or an *expert* in formal methods. An *expert* safety engineer states that “*the degree of difficulty heavily depends on the nature of the formalism: Spotting an error in a boolean formula is significantly easier than identifying erroneous temporal specification, which is again significantly easier compared to spotting erroneous specification for continuous behavior*”.

Summary. Predominantly, participants answer that identification of inconsistent specifications is *hard*. Free-text answers agree that the effort for identifying inconsistent specifications highly depends on the complexity of specifications and systems.

6.3.2 Understanding inconsistency in formal specifications

The question *Q10* (Table 1) is used to collect whether understanding of the actual inconsistency in the identified inconsistent specifications is *easier* or *harder*. The answer scale used to collect the response is again *LS3* and a free text field. Responses are aggregated in Table 6, where 39 participants provided a rating, while two participants indicated *no opinion*.

The result is comparable to the result obtained for identifying inconsistent specification in Section 6.3.1. The predominant number of 18 participants an-

Table 6: Understanding inconsistent formal specification.

Agreement	Extremely Hard	Hard	Slightly Hard	Neither Hard nor Easy	Slightly Easy	Easy	Extremely Easy	No Opinion
Count	6 (15%)	18 (44%)	6 (15%)	5 (12%)	2 (5%)	1 (2%)	1 (2%)	2 (5%)

swers understanding inconsistency in the formal specifications is *hard*. Overall 77% of participants responded at least *slightly hard*. 13% answer *neither hard nor easy* and perceive it *easier*.

Three participants who answer *neither hard nor easy* highlight that the effort of understanding the inconsistency in formal specifications depends on the complexity of specification and the system. A *verification expert* with the response *hard* states that “*the key question is: whether the model or the specification is wrong i. e., finding inconsistencies can often not only be done on the level of just the specification*”.

Understanding inconsistency in formal specifications based on the participants' knowledge in formal methods. Figure 8 depicts the responses gathered for understanding inconsistency in formal specifications together with the participants knowledge in formal methods (*cf.* Section 6.1.2). The majority of participants from *novices* to *experts* perceive understanding of inconsistencies to be *harder*. Particularly, this view is shared by six of eight *novices*, seven of 11 *advance beginners*, four of seven *competents*, all four *proficients*, and seven of 11 *experts*.

One of the *advanced beginners* explains a common industrial issue: “*understanding can be hard especially when the specifications have multiple authors, each holding a slightly different view on the object to be specified*”. A *system/software architect* who rates their knowledge in formal methods as *expert*, states that “*formal proofs typically lead to the source of the error. Counterexamples in model checkers also help a lot*”. On contrary to this statement a *systems engineer* who rates their knowledge in formal methods as *expert* highlights that it depends on the particular inconsistency, *e. g.*, if it is real-time

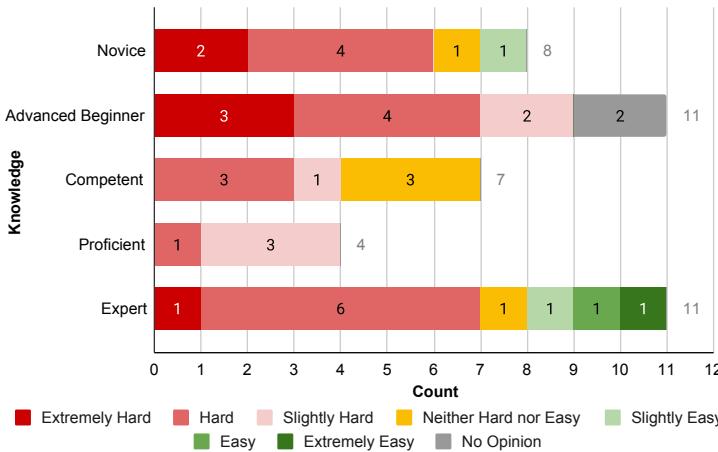


Figure 8: Results for the complexity in identifying inconsistency in formal specifications grouped by the participants' knowledge in formal methods.

inconsistency that only occurs after several cycles, then it is difficult to find even with the help of counterexamples.

Summary. Similar to the result of identifying inconsistent specification in Section 6.3.2, a majority of participants (77%) value understanding of the inconsistency as harder. In the free-text responses, participants indicate that the effort to understand an inconsistency depends on the complexity of the specification and system. Further, some of the responses highlight the use of tools like model checkers to support identification and understanding of inconsistencies.

6.3.3 Time Taken to Identify Inconsistent Formal Specifications

With question *Q11* (Table 1), we consider the time that it takes to identify inconsistent specifications. The answers follow scale *LS4* (Table 2) and allow for a free-text comment. Table 7 represents the response collected from 37 participants as four participants responded *no opinion*.

A majority of participants, 18 of them (44% of all participants) answer that the time taken to identify inconsistent specifications is *slower* (*slightly slow*, *slow*, and *extremely slow*). Further, 12 participants (29%) answer that the time taken to identify inconsistent specifications is *neither fast nor slow*, with six participants stating that it depends on the complexity and number of specifications.

A *system/software architect* answering with *slightly slow* highlights that “*the time it takes is the time taken to understand requirements*”. Further, a *verification expert* states that in one of their projects it took a couple of hours to formalize the system model and specifications, but only a few seconds to then run the model checking and perform verification and further optimization.

Seven participants (17%) answer that the time taken to identify inconsistent specifications is *faster* (*slightly fast*, *fast*, and *extremely fast*). Three of them answer that by using a model checker, the time taken to identify inconsistent specifications is reduced.

Time taken to identify inconsistent formal specifications grouped by the participants’ knowledge in formal methods. Figure 9 presents the results of time taken to identify inconsistent formal specifications grouped by the participants’ knowledge (cf. Section 6.1.2). The majority of participants who rate their knowledge as *novice*, *advanced beginner*, *competent*, and *expert* answer that the time taken to identify inconsistent specifications is *slower* or *neither fast nor slow*. Five of seven participants who answer *faster* rated themselves

Table 7: Time taken to identify inconsistent formal specifications.

Agreement	Extremely Fast	Fast	Slightly Fast	Neither Fast nor Slow	Slightly Slow	Slow	Extremely Slow	No Opinion
Count	0	3 (7%)	4 (10%)	12 (29%)	4 (10%)	11 (27%)	3 (7%)	4 (10%)

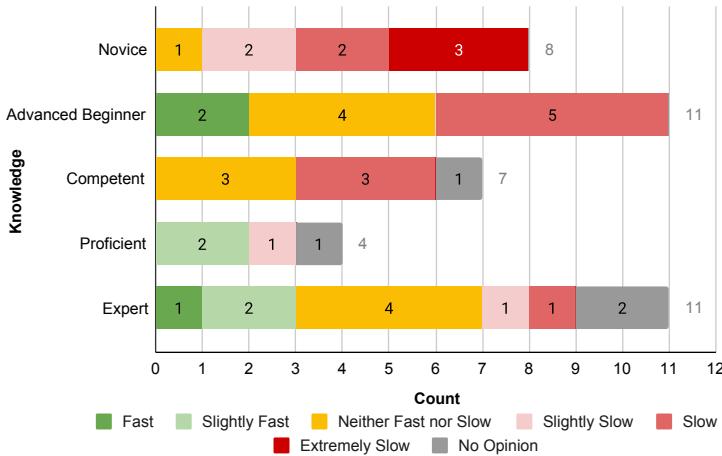


Figure 9: Results for the time taken to identify inconsistent formal specifications grouped by the participants' knowledge in formal methods.

as *proficient* and *experts*. The majority answers of about 37% of all participants respond that using formal verification makes *very probably* a system safer while 34% vote for *definitely* safer.

Summary. In total, 44% of the participants answer that identifying inconsistent formal specifications is *slower*. In fact, 44% is predominant here because 29% of them have the neutral response answering *neither fast nor slow* and only 17% of the participants answer to be *faster*. Free-text responses indicate that the time it takes depends on the kind of specifications and the complexity of the system and specifications. This is similar to the responses received for identifying inconsistent specifications in Section 6.3.1 and Section 6.3.2.

6.3.4 Challenges to Identify Inconsistent Specifications

With question *Q12* (Table 1), we collect challenges in identifying inconsistent specification based on free-text responses. The responses are summarized based on the participants' designations.

Architects and Experts. Most of the *system/software architects*, *safety/security experts*, and *verification experts* highlight two general challenges: (1) architectural models and system requirements are often incomplete in industry, and (2) understanding the semantics and formal notations. A notable statement from a *system/software architect* is that “*if the specification has many items (which is very common in the industry), checking them one-by-one by human is time-consuming and error-prone. Furthermore, if the specification is written in natural language, how to interpret it in an objective way can also be a question*”. Further, a *verification expert* states that it is challenge to “*understand*

the intended semantics of the specifications at scale”. Another notable statement from a *safety/security expert* is that “*the prime challenge is to identify inconsistency and then to get acceptance to the amount of inconsistencies [that can be tolerated]*”.

Safety Managers/Engineers. Two main challenges listed by *safety managers and engineer* are: (1) understanding formal specification and (2) understanding verification results. A *safety engineer* rated as *expert* states that “*some notions (e.g., LTL formulas) are hard to understand. Often you just think you understood correctly what they mean while using them the wrong way. Furthermore, there might be corner cases in the processes to be described which make your life particularly hard, e.g., specific startup behavior of systems. Even simple formulas tend to become huge quickly, which makes it pretty hard to focus*”. Furthermore, a statement from a *novice* is, that “*it is not really intuitive to understand the dependencies between the formal specifications – especially if you investigate a larger set of specifications*”.

Systems Engineers. The main challenge mentioned by *systems engineers* are formalizing specifications. A *systems engineer* rated as *expert* states that “*the main challenge to identify inconsistent formal specifications is that specifications have to be formalized first. There are only few engineers at Bosch who want to do that, even semi formal requirements patterns are seldom used*”. Further, a challenge mentioned by a *systems engineer* rated as *competent* is “*insufficient granularity of high-level system behavior, thus difficult to perform verification*”. In addition, a general challenge to identify inconsistent specification is that “*often hundreds of cases/situations have to be considered while only a few are inconsistent*”.

Research Engineers and Verification Engineers. Two challenges identified by *research engineers* and *verification engineers* are: (1) complexity in using appropriate tools, and (2) understanding the verification results. A *research engineer* rated as *proficient* states that the “*large size and complexity of the specification might not allow one to debug it manually. When the formal specification is written in an expressive language and is very complex in general, automatic method for formal analysis might not scale either*”. Further, a *verification engineer* highlights that even if a model checker supports to identify an inconsistent specification, the time taken to perform verification for large system is huge. Regarding explainability, a *research engineer* states that inconsistencies detected by the verification tools are cryptic and thus, require additional support to derive a useful explanation.

Summary. From the collected responses, the four different challenges to identify inconsistent specification are: (1) verification performed with incomplete models, (2) understanding formal semantics, notations, and specifications, (3) complexity in using verification tools, and (4) understanding of verification results by domain experts.

6.3.5 Methods used for Verification

As discussed in Section 6.3.2, several participants mentioned the use of verification methods to identify inconsistent specifications and understanding the inconsistencies. With *Q13* (Table 1), we collect the used verification methods with a free-text field. Several participants mentioned multiple methods. All responses are clustered into five different methods shown in Table 8.

The predominantly used methods are *model checking* with 18 participants and *manual inspection/review* with ten participants. *Manual inspection/review* is the only purely manual method, the other methods like *model checking*, *simulation*, *reasoner*, *contract-based design* are (semi-)automated methods, supported by appropriate tools. Five participants indicate that model checkers support overcoming the manual inspection to find inconsistency.

Methods used for verification based on designation. Figure 10 represents the different verification methods together with participants' designation. *Model checking* is used by at least one participant of each designation. *Manual inspection/review* is mostly mentioned by *systems engineers* and *safety managers/engineers*, *simulation* mostly by *systems engineers* and *verification experts*, *reasoners* by *research engineers*, and *contract-based design* mostly by *safety managers/engineers*.

Verification methods and the complexity of identifying inconsistent specifications. Figure 11 depicts the different verification methods together with the

Table 8: Results for the used verification methods.

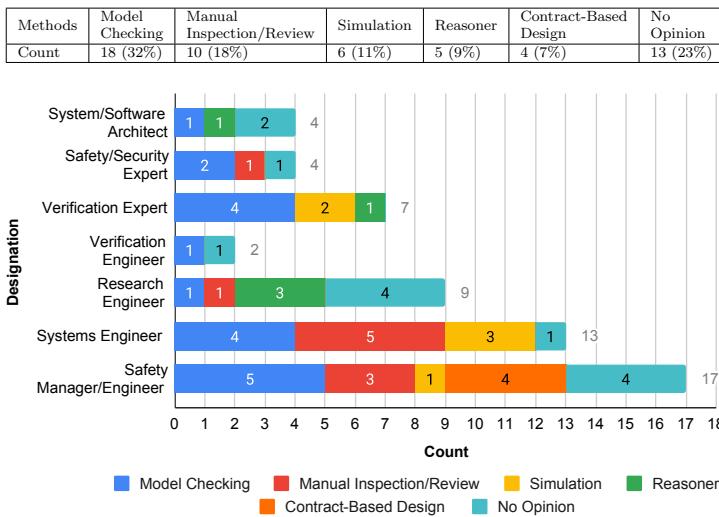


Figure 10: Methods used by participants to identify inconsistent formal specifications grouped by designation.

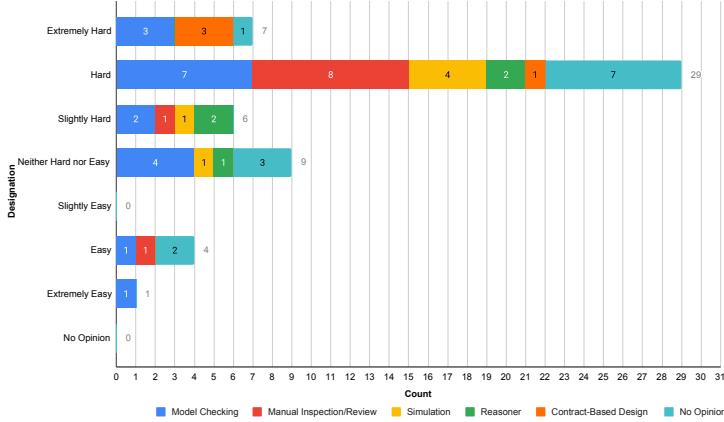


Figure 11: Methods used to identify inconsistent formal specifications along with complexity of identifying inconsistent specifications.

results obtained for identifying inconsistent formal specifications in Section 6.3.1. Since the results for both identifying inconsistent specification (*cf.* Section 6.3.1) and understanding inconsistent specification (*cf.* Section 6.3.2) are similar, we choose to discuss the relation only with results of identifying inconsistent specification.

As a majority of the participants (21 of 41 participants) have answered that identifying inconsistent formal specifications is *hard*, they mentioned each of the listed methods at least once. To be more precise, among 21 participants who answer *hard* for identifying inconsistent specification, eight participants use *manual inspection/review* while seven of them use *model checking*. Additionally, participants who answer *extremely hard* are found to use *model checking* and *contract-based design*. A participant who rated herself/himself as *proficient* and classify the problem as *hard* state that the correct usage of verification tools is mostly an error-prone way. Furthermore, an *expert* states: “*If you see UML Models Rhapsody/EA as formal models: typically, they are not complete and therefore model checker do not support well*”. On other hand, participants classifying the problem as *easier* are found to use *model checking* and *manual inspection/review*. An *advanced beginner* classifying the problem as *easy* by using *manual inspection/review* highlights that “*by using the modern requirement engineering tools traceability can be maintained within dependent specification and corresponding architectural model*”.

Summary. From the collected responses, five different verification methods are used to identify inconsistent specifications: *model checking*, *manual inspection/review*, *simulation*, *reasoner*, and *contract-based design*. Among these, the predominantly used methods are *model checking* and *manual inspection/review*, mentioned by 18 participants and ten participants respectively.

6.4 Refinement of Specifications

The early stages in the V-model (Weber 2009) are about refining the top-level requirements and associate them to system components and sub-components. With questions *Q14* and *Q15* (Table 1), we investigate whether maintaining the consistency of refined specifications with a system architecture and verifying the refinement consistency are *harder* or *easier*.

6.4.1 Consistency of Refined Specifications and System Architectures

With question *Q14* (Table 1), we collect the responses whether maintaining consistency of the refined specifications is *harder* or *easier* following the answer scale *LS3* in Table 2. Twelve out of 41 participants state that they have *no opinion* and thus, the responses received from 29 participants are shown in Table 9. The predominant number of participants answer that maintaining consistency is *hard* and *slightly hard*, with an exact count of nine participants (31% of responded participants) each. Overall, most participants (20 participants, 69% of all participants) answer that maintaining consistency is *harder*, eight participants (28%) answer with *easier*, and one participant (3%) with *neither hard nor easy*.

Maintaining refinement consistency grouped by the participants' experience in formal methods. Figure 12 depicts the results of maintaining refinement consistency between the specifications and architecture considering the participants' knowledge in formal methods (*cf.* Section 6.1.2). Participants who rate the difficulty as *extremely hard* are *novices* and *advanced beginners* while those who answer *extremely easy* are designated as an *expert*. Since the majority of participants rate the difficulty as *harder* (*cf.* Table 9), all the categories from *novice* to *expert* have higher count for *harder* than *easier*. An *advance beginner* who rates the problem as *hard* highlights: “*It depends on the size and complexity of the System-Architecture; in L4 Sensor-set architectures it[']s hard*”. Additionally, an *expert* states that performing a manual review with natural language requirements against the requirements of the higher abstraction level in a contract-based design is *hard*. From all of the categories except of *proficient*, at least one participant answers that maintaining inconsistent specifications is *easier*. An *advanced beginner* whose rates the problem as *easy* mentions: “*I assume I formulated a formal requirement for the parent component, and I have to define formal requirements for sub-components, I think that is straight forward. If somebody else wrote the formal requirement of the parent component, it becomes harder since I need to understand what is expressed there first*”.

Table 9: Results for the difficulty of maintaining the consistency when refining formal requirements for sub-components of a system architecture.

Agreement	Extremely Hard	Hard	Slightly Hard	Neither Hard nor Easy	Slightly Easy	Easy	Extremely Easy	No Opinion
Count	2 (5%)	9 (22%)	9 (22%)	1 (2%)	5 (13%)	2 (5%)	1 (2%)	12 (29%)

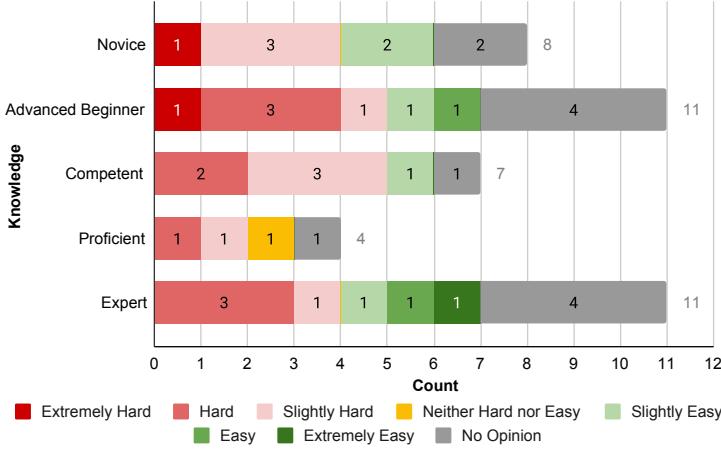


Figure 12: Results for the difficulty of maintaining refinement consistency grouped by the participants' knowledge in formal methods.

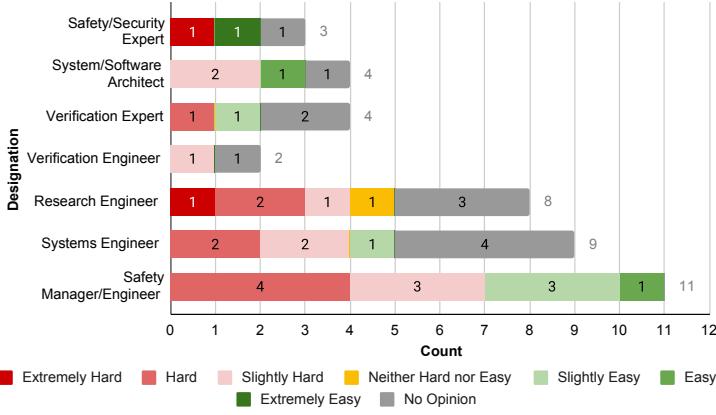


Figure 13: Results for the difficulty of maintaining refinement consistency grouped by the designations of the participants.

Maintaining refinement consistency grouped by the participants' designations. Figure 13 depicts the results for the difficulty of maintaining refinement consistency between the specifications and architecture grouped by the participants' designations (*cf.* Section 6.1.3). Focusing on valid responses (excluding *no opinion*) from *system/software architect*, *safety/security expert*, and *verification expert*, four participants answer *harder* and three participants answer *easier*. Notably, among two *system/software architect*, one answers *extremely easy* and the another one answers *extremely hard*. The participants who answer *extremely easy* state that using model-based development it is easy to maintain the consistency with natural language requirements. Additionally, according to a *system/software architect* who answers *slightly hard*, maintaining refine-

ment consistency “[s]cales with number of requirements. With graphical models like state machines one can keep complexity under control”. Participants from the remaining designations like *systems engineer*, *verification engineer*, *safety manager/engineer*, and *research engineer*, predominantly rated the problem as *harder*. A *safety manager/engineer* classifying the problem as *slightly hard* states: “*To maintain the consistency, we need to refine all software development phases results from requirements till test*”.

Summary. Among all of the 29 participants who answered this question, 20 participants (69%) vote that maintaining consistency is *harder*. From the free-text field responses, the majority of the participants finds that maintaining consistency is *harder* when the system gets complex. Furthermore, a notable response is that model-based system development could ease maintaining consistency.

6.4.2 Verification of the Refinement Consistency

With question *Q15* (Table 1), we collect the responses whether verifying refined specifications is *harder* or *easier*. Possible responses follow answer scale *LS3* in Table 2. Among the 41 participants, eleven answer to have *no opinion* while the remaining 30 participants rate the difficulty of the verification problem. The overall result shown in Table 10 is similar to the responses of *Q14* about maintaining consistency discussed in Section 6.4.1. The majority of 24 participants (80% of responded participants) answer that verifying refined specifications is *harder*. Further four participants answer it to be *easier* and two participants answer, *neither hard nor easy*. Notably, among the 24 participants who answer *harder*, 16 (66%) answer *hard* which takes the predominant count of all the given options.

Verifying refinement consistency grouped by the participants’ knowledge in formal methods. Figure 14 depicts the results of rating the difficulty of verifying refinement consistency grouped by the participants’ knowledge in formal methods (Section 6.1.2). None of the *novice* and *proficient* participants rate the problem of verifying refinement consistency as *easier*. A *proficient* participant mentions: “*You can only rely on provisioned tools (model checker)*”. Among the rest, at least one participant answers that verifying refinement consistency is *easier*, however, the majority of the answers rate it to be *harder*. According to an *advanced beginner* rating the problem as *hard*, the difficulty depends on the formalization of requirements: “*If you mean non-formal requirements, I*

Table 10: Difficulty of checking consistency when refining formal requirements for sub-components of a system architecture

Agreement	Extremely Hard	Hard	Slightly Hard	Neither Hard nor Easy	Slightly Easy	Easy	Extremely Easy	No Opinion
Count	2 (5%)	16 (39%)	6 (15%)	2 (5%)	2 (5%)	1 (2%)	1 (2%)	11 (27%)

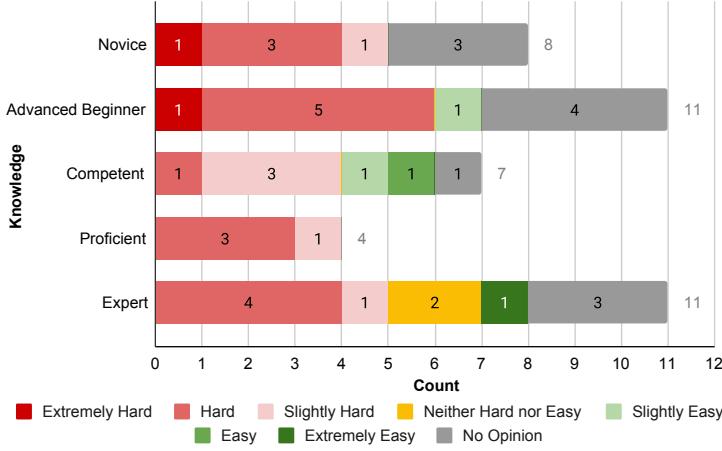


Figure 14: Results for the difficulty of verifying refinement consistency grouped by the participants' experience in formal methods.

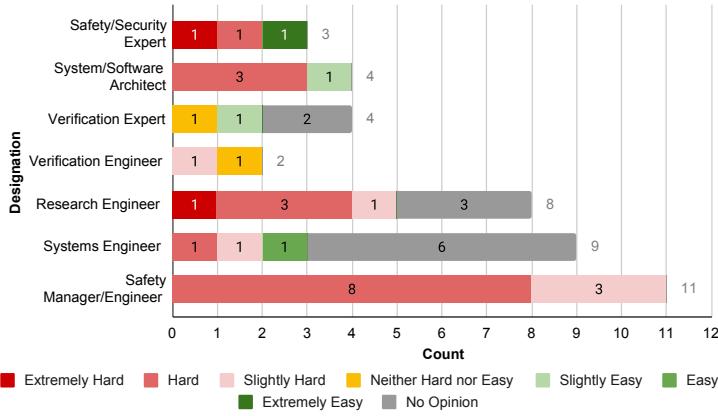


Figure 15: Results for the difficulty of verifying refinement consistency grouped by the participants' designations.

think checking the consistency is extremely hard. If you mean formal requirements but linked to components, I think it is slightly hard. However, links to components help to understand the relationship between variables”.

Verifying refinement consistency grouped by the participants' designations. Figure 15 depicts the results of rating the difficulty of verifying refinement consistency grouped by participants' designations (Section 6.1.3). Focusing on the designation of a *safety/security expert*, one participant answers that verifying refinement consistency is *extremely hard* while another participant answers it to be *extremely easy*. One of the *safety/security expert* states that “[e]specially when the requirements to be checked are distributed, without tool

support, checking can be extremely tedious”. With the designations of a verification engineer, safety manager/engineer, and research engineer as exceptions, all other designations have at least one participant rating the difficulty as *easier*. A *verification engineer* rating the problem as *neither hard nor easy* mentions: “*Ideally, the formal requirements should be specified in a way that they can be automatically checked by some tool – then it is easy*”. Additionally, a *systems engineer* rating the problem as *easy* states that “[*it is easy as long as the requirements can be functionally separated. If different requirements have impact on a state behavior or contradicting safety goals, it is hard*”.

Summary. Among the 30 participants who provide a response other than having *no opinion*, 80% perceive the verification of refinement consistency as a *harder* problem. Furthermore, most participants who rated the problem as *easier* highlight that the effort of verifying refinement consistency could be reduced by using well-formalized specifications and verification tools.

6.5 Formal Verification Focusing on Safety

With questions *Q16* to *Q18*, we collect the participants’ opinion on whether formal verification could support safety analysis and make a system safer.

6.5.1 Using Formal Verification to Make Systems Safer

Particularly, with question *Q16* we investigate the opinion of participants on whether using formal verification could make a system safer (response according to scale *LS5* in Table 2 plus free-text comments). Aggregated responses are shown in Table 11. A predominant number of participants answers *definitely* (14 participants, 34%) and *very probably* (15 participants, 37%).

Most free-text comments collected by participants who answer *definitely*, *very probably*, or *probably* highlight that formal verification is required to tackle the increasing system complexity and shorten development time. A *research engineer*, whose answer is *definitely*, states that “*it can provide yet another level of surety about the safety of systems. The more sure we are the better it is. The human brain tries to save energy. As a human, we may overlook a lot of details when it comes to habitual/routine work. This is where formal methods can provide further surety about safer systems.*” A *safety engineer*, whose answer is *very probably*, highlights a further challenge in industry: “*Personally, I think it is useful if the environment is open to this (e.g., interested in the results and willing to spend resources on it) and you have the right well-trained people working on it. However, I’ve seen these ideas failing too many times*

Table 11: Results on whether using formal verification makes systems safer.

Likelihood	Definitely	Very Probably	Probably	Neither Probably nor Possibly	Possibly	Probably Not	Definitely Not	No Opinion
Count	14 (34%)	15 (37%)	6 (15%)	1 (2%)	4 (10%)	0	1 (2%)	0

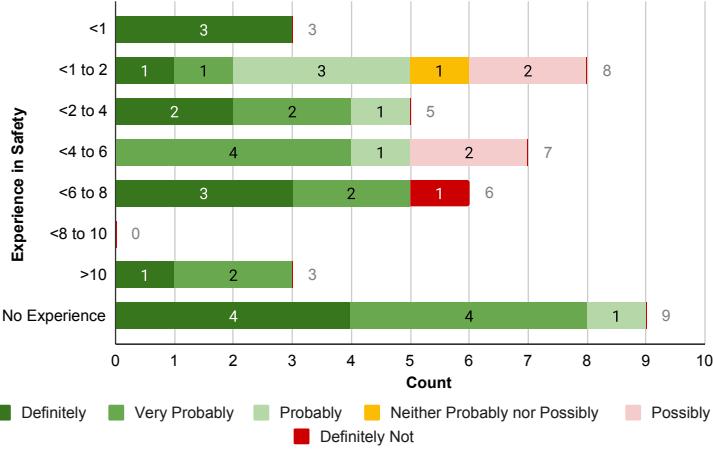


Figure 16: Results on whether using formal verification makes systems safer grouped by the participants' industrial experience in safety.

in industry. I conclude that in most cases using formal verification is not yet ready for production.”

Relation to participants' experience in safety. Figure 16 depicts the results of using formal verification to make systems safer based on the participants' industrial experience in safety. Nine participants with *no experience* present their view that using formal verification makes *definitely*, *very probably*, or *probably* the systems safer. All of the participants with <1 year experience answer that using formal verification could make the system *definitely* safer. An example statement by a *systems engineer* without safety experience states that “[g]aps and errors can be found earlier, easier and with higher reliability.” Considering the experience levels from 1 to <2 years and up to >10 years, the majority of participants answer with *very probably*. A *safety manager/engineer* with 2 to <4 years of experience who answered with *very probably* mentions that formal verification “can help to make system consistent and reduce the human error.” Furthermore, another *safety manager/engineer* strengthens this view: “With higher system and organizational complexity combined with shorter development time, the need for more formal methods is increasing.”

Focusing on the answers *possibly* and *definitely not*, two participants either with 1 to <2 or 4 to <6 years of experience answer *possibly* while one participant with 6 to <8 years of experience answers *definitely not*. A *safety manager/engineer*, whose answer is *definitely not*, states the following: “We need to distinguish between two terms: Reliability and Safety. The formal verification are used to define the failures /Errors in software but not necessary that the all errors are safety-critical”. Finally, a *verification engineer*, whose answer is *possibly*, highlights that formal verification “could potentially make them safer, but I think the effort is probably too high in relation to the benefit.”

Summary. The majority answers of about 37% of all participants respond that using formal verification makes *very probably* a system safer while 34% vote for *definitely* safer. Most of the participants highlight that using automated formal verification methods could reduce a lot of manual work. However, a considerable number of participants also highlight that without proper training it is hard to imagine using formal verification in industry.

6.5.2 Formal Verification as an Add-on

With question *Q17*, we collect the participants' opinion whether formal verification could be a meaningful addition to the functional safety methods to ensure safety (possible answers follow scale *LS5* in Table 2 and allow free-text comments). Among the 41 participants, 39 have answered this question. Results are shown in Table 12. From these 39 participants, 19 (49%) answer that formal verification could *definitely* be an add-on to functional safety methods, and eleven participants (28%) consider this as *very probably*. A *safety manager/engineer*, whose answer is *very probably*, states that “*formal verification methods probably can [be] used to specify and verify the functional safety related properties of the system e. g., time related issues such as Fault-Tolerance Time Intervals etc.*” The answers to question *Q16* (*cf.* Section 6.5.1) correlates with the results shown in Table 12.

Table 12: Results for whether using formal verification could be a meaningful add-on to functional safety methods to ensure safety.

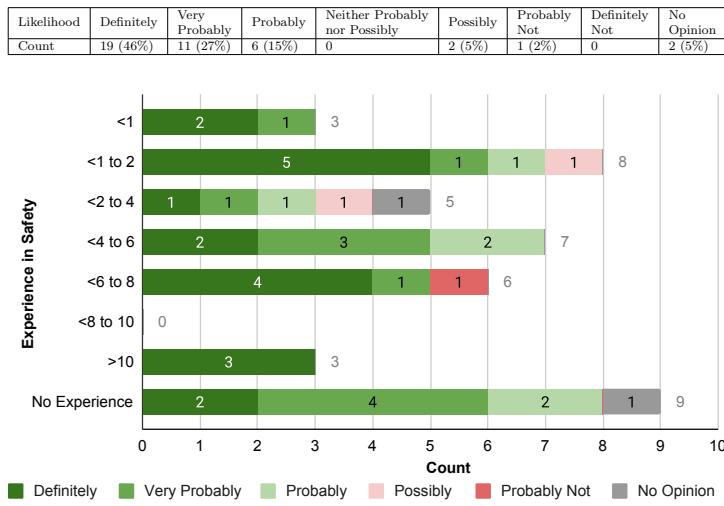


Figure 17: Results for whether using formal verification could be a meaningful add-on to the functional safety methods grouped by the participants' experience in safety.

Relation to participants' safety experience. Figure 17 depicts the results grouped by the participants' experience in safety (*cf.* Section 6.1.1). The predominant number of participants having an experience of 6 to <8 years and > 10 years answer *definitely*. A *safety manager/engineer* with 6 to <8 years of experience states: “*To me, formal verification is already a method to achieve functional safety. It's just not applied widely. Formal verification itself cannot ensure safety, it can just contribute.*” Furthermore, a *system/software architect* with > 10 years of experience mentions the challenge that mostly formal verification is not scalable, and very costly. Notably, among all participants, only one participant who answers with *probably not* and having between six and eight years of experience states the following: “*We have two terms: functional safety addresses E/E failures in the systems and SOTIF (safety of intended functionality) addresses the safety issues in absence of E/E failures. Therefore, to ensure safety, we need to reduce the hazards which are related to E/E failures + Performance limitations.*”

Summary. 19 participants (49% of the responding participants) deem formal verification *definitely* to be an add-on to classical functional safety methods. From the free-text responses, most participants state that formal verification together with functional safety methods could address more system safety issues.

6.5.3 Benefit of Identifying Inconsistent Specifications

With question Q18, we collect the participants' opinion whether identifying inconsistent formal specifications is beneficial for a safety analysis (possible answers are defined by scale LS5 in Table 2 and allow free-text comments). Forty participants provided answers and one participant had *no opinion* (Table 13). The majority of 21 participants (53%) rates identifying inconsistent formal specifications *definitely* beneficial for a safety analysis. The second most-called answer with 15 votes (38%) is *very probably*. A *system/software architect* answering *definitely* states that “*it is beneficial because: (1) it saves time and effort trying to resolve the inconsistencies in the future, and (2) avoids possible security threat due to the vagueness in the specification.*” A research engineer highlights that “*naturally, safety critical application which rely on inconsistent formal specifications can never be guaranteed to function reliably.*”

Relation to participants' experience in safety. Figure 18 depicts the results in relation to the participants' experience in safety. A majority of participants between 0 to 8 years and >10 years vote for *definitely*. A *research engineer* answering *definitely* and with 2 to <4 years of experience states that “*if formal methods were to be used extensively in the production domain, it would be very important because it is the starting point of the analysis and will have great influence on the results.*” Further, a *safety manager/engineer* with 6 to <8 years experience mentions that “[identifying] inconsistent specification is definitely beneficial, not so much for the safety analysis but for the safety of

Table 13: Results for whether identifying inconsistent specifications is beneficial for a safety analysis.



Figure 18: Results for whether identifying inconsistent specifications is beneficial for a safety analysis, grouped by the participants’ experience in safety.

the product itself. Also, it is not limited to the safety but general performance of the product.” Looking at all benefits, a safety manager/engineer answering definitely and with 4 to <6 years experience mentions that “it is indeed very helpful. However, the harder part is to write correct (partial) specifications in the first place. If you fail to do so, there is no benefit from making your (flawed) specification consistent.”

Summary. 53% of the participants who provided responses see that identifying inconsistent formal specifications could *definitely* be beneficial for a safety analysis. Most of the free-text comments highlight that identifying inconsistencies could save time and effort very early and improve system safety.

6.6 Using Formal Verification

The questions *Q8*, *Q19*, and *Q20* (Table 1) collect the participants’ opinions on using formal verification in general, imagining using formal methods if understanding formal notations is eased, and using formal methods in real-world development. The responses to these questions are discussed in the following.

6.6.1 Opinions on Using Formal Verification in General

Question *Q8* collects opinions of the participants on using formal verification with a free-text field. The responses are categorized based on the participants’ designations.

System/Software Architects and Safety/Security Experts. A *system/software architect* with their knowledge rated as *expert* advises to “*not try to model your whole system with formal methods. Model relevant aspects and check them.*” In addition, a *system/software architect* as an *advanced beginner* states, that “[*formal methods*] are helpful and can bring great help if used properly. However, there could be some effort initially for translating existing specifications into the representation or notation used by formal verification tools.” Further two *system/software architects* mention that with the required knowledge it is easy to use formal tools but still the question remains whether it scales to industrial systems.

Responses from *safety/security experts* show that they are interested in different perspectives. For example, a *safety/security expert* who is an *advanced beginner* states the expectation that “*one very promising solution is needed to compose/decompose safety requirements for component based development.*” Further, an *expert* highlights that they only “*see rare use cases, since the benefit most often does not outweigh the costs (yet). This is, however, different when the formal verification is practically hidden and specification is intuitive. See for example type systems for programming languages checked by compilers without support from the user.*”

Verification Experts and Verification Engineers. The responses received by verification experts and engineers indicate that they are mainly interested in verifying system architectures and requirements. A *verification expert* highlights that “*if hard statements about correctness are required, [*formal methods*] are inevitable if they can be used. In addition, they are very good at finding corner cases that are hard to find (e.g., transient errors resulting from concurrency).*” Further, a *verification engineer* states that formal methods “*should be used for highly critical parts (only). I'm not sure if it is useful to apply it on higher levels (e.g. system architecture). That might help to get a consistent picture on that level, but will probably not help to ultimately build a safe system, as the properties can usually not be checked on implementation level.*”

Safety Managers/Engineers, Systems Engineers, and Research Engineers. Most of the *safety managers/engineer* and *systems engineer* are interested to use formal methods, but the complexity of using and understanding formal tools stands as a barrier. For example, a *safety engineer* states that formal methods “*can bring a huge benefit to safety engineering, but some fundamental challenges remain (competency of engineers applying formal methods and tools).*” Furthermore, according to the statement by a *systems engineer*, “*formal verification is powerful, but to get the acceptance we have to keep the formal stuff away from the users. E.g., Astree, Polyspace, QA-C all find defects, that are nearly impossible to detect by hand written tests. Astree and Polyspace find more than QA-C, still, many projects use QA-C, as it is extremely easy to use even for people without knowledge in formal methods.*” In addition, a research engineer highlights a crucial challenge: “*I do think it makes sense to apply formal verification especially on highly complex and*

safety-critical system, but the hurdle might be very high for its wide application. We will require not only safety engineers to understand the method, notation, syntax, etc. but also e.g., system designers, software developers. If external certification authority is involved in the certification process, it might be an additional challenge to present the safety case (unless they are experts in formal verification).”

Summary. The majority of collected responses are very positive in using formal verification to improve the system safety and design. But, on other hand understanding and scalability of verification (tools) remain as obstacles.

6.6.2 Opinions on Using Formal Verification if Understanding of Formal Notations is Eased

Question *Q19* collects the opinion of participants on whether making formal notations more understandable could improve the usage of formal methods. The possible answers follow scale *LS5* in Table 2 and can be extended with free-text comments. All of the 41 participants provided responses, which are shown in Table 14. The majority of participants (23, i.e., 56%) answer that making formal notations more understandable could *definitely* improve the usage of formal methods. Eleven participants (27%) estimate that it will *very probably* provide an improvement. A verification expert answering *definitely* highlights that “*increasing understandability of formal specifications is key to bringing them into a more wide-spread use. See for example Gladisch et al. (2019) lessons learned on SBT and the STL specifications.*” Finally, four participants expect that easing understanding of notations will *probably* improve the usage of formal verification while and one participant each votes for *neither probably nor possibly*, *possibly*, and *probably not*.

Understanding of formal notations is made easier based on formal methods knowledge. Figure 19 depicts the results of whether making formal notations more understandable could improve the usage of formal methods considering the participants’ knowledge in formal methods (*cf.* Section 6.1.2). Except of *novices*, the majority of participants rated from *advanced beginners* to *experts* expect a *definite* improvement. A highlighting statement from a *safety engineer* with knowledge rated as an *advanced beginner* states that, however, “*it is not just the decision of the safety team, also the System, SW and maybe the HW experts and also testers needs to understand it to make the investment in formal methods reasonable.*” In addition, a *safety engineer* states the

Table 14: Results of using formal methods if understanding of formal notations is made easier.

Likelihood	Definitely	Very Probably	Probably	Neither Probably nor Possibly	Possibly	Probably Not	Definitely Not	No Opinion
Count	23 (57%)	11 (27%)	4 (10%)	1 (2%)	1 (2%)	1 (2%)	0	0

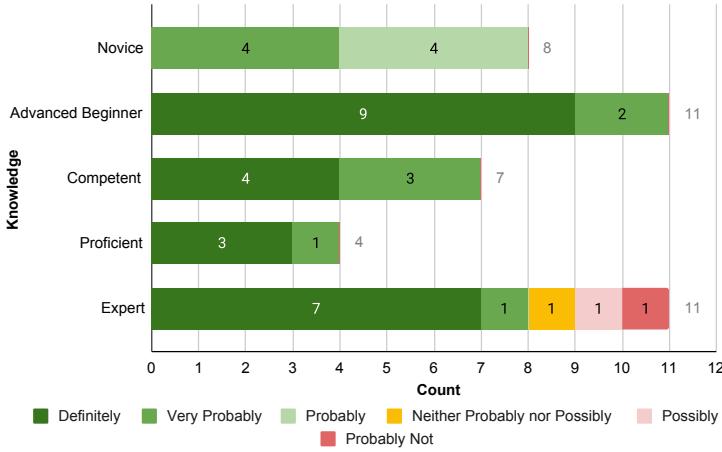


Figure 19: Results for using formal methods if understanding of formal notations is made easier, grouped by the participants knowledge in formal methods.

following: “*To me, understandability is one of the main reasons that it [formal verification] is not applied, besides the huge initial effort (modeling and specifying a system formally).*”

From participants with *expert* knowledge in formal methods, one participant each answers with *neither probably nor possibly, possibly, and probably not*. From the collected free-text comments, most participants consider the formalization of system models and specifications from informal systems descriptions and requirements as challenging. For example, a participant answering with *neither probably nor possibly* highlights that “*the problem is not related to method itself but it is related to how you use the method and what are the inputs, and how you understand the system architecture and artifacts. Also, how you apply the method and so on.*” A further participant answering with *possibly* mentions that “*the formal notation has to be very simple, it has to [be] comprehensible including small details, and it has to be sufficiently flexible so it can be used in unforeseeable use cases.*” A participant answering with *probably not* states that “*specifying a formal model is hard. If you can manage that, you can also manage the notation.*”

Summary. 56% of all participants think that making formal notations more understandable could *definitely* improve the usage of formal methods. However, understanding and formalizing systems and requirements in the first place remains still a barrier to use formal methods.

6.6.3 Opinions on Using Formal Methods in Real-World Development

With question *Q20*, we collect the opinions of participants on whether formal methods are usable in real-world development processes. Possible response

follow scale *LS5* in Table 2 and could be extended by free-text comments. While one participant has *no opinion*, the remaining 40 participants provide responses shown in Table 15. Predominantly, the answers are positive with most participants (13, 33%) answering with *probably*, 11 participants (28%) with *definitely*, and ten participants (25%) with *very probably*. Only one participant each answers with *neither probably nor possibly* and *definitely not*.

Usability in real-world development processes based on formal methods knowledge. Figure 20 depicts the results of whether formal methods are usable in real-world development processes considering the participants' knowledge in formal methods (*cf.* Section 6.1.2). Most answers received as free-text comments highlight the interest to use formal methods, however, understanding and familiarizing with the notations and tools still need to be improved. From the majority of 13 answers selecting *probably*, seven participants rated their knowledge as an *advanced beginner*. Such a participant highlights that “*the huge WHY NOT in my opinion is the frontloading/enabling. Application of formal methods usually requires more time in the beginning to become familiar with the formalization method (and maybe tools). Often, representation is not intuitive on the first glance and requires a few people who actually ‘dig themselves into the problem’. This makes it unattractive for fast applications/scouting. As soon as you started with another type of analysis in the beginning, the technical debt still changing to a more formal method becomes higher and higher. Thus, in my opinion, formal methods can be very helpful*

Table 15: Results of using formal methods in real-world development.

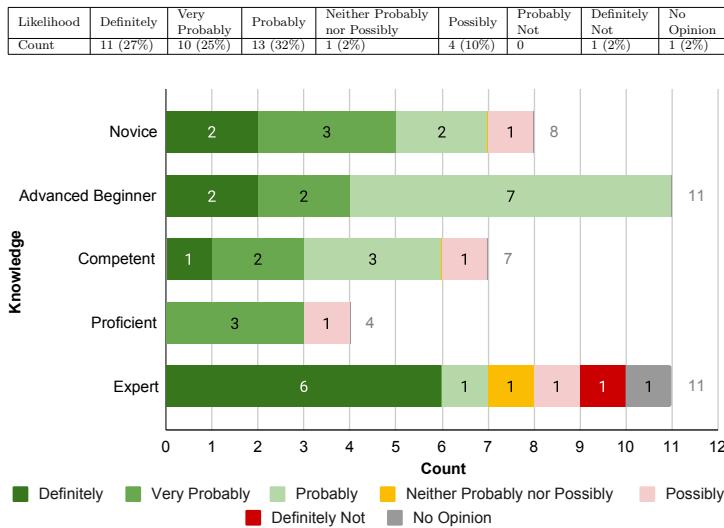


Figure 20: Results of whether formal methods are usable in real-world development processes grouped by the participants' knowledge in formal methods.

but require a good visualization and explanation as – in the beginning – these methods are often an investment of time for a later quality return.” Furthermore, notably, six of the 11 participants answering with *definitely* rated their knowledge as an *expert*. For instance, such participants see the benefits but also the prerequisites of using formal methods: “[Formal methods] are usable and can provide a huge benefit. However, the necessary competences have to be created, the management has to support this.”

Usability in real-world development processes based on designations. Figure 21 depicts the results of whether formal methods are usable in real-world development processes grouped by participants’ designations (Section 6.1.3). Among 11 participants answering with *definitely*, eight participants are *system/software architects*, *systems engineers*, and *research engineers*. An example statement from a *systems engineer* is that “formal methods are already in use, e. g., in tools like *Astreee*, *Polyspace*, etc. This only works, if we try to make it as easy for the user as possible. I don’t see people formalizing requirements but, I think if we lower the hurdle this might also change on the left side of the V.” Similarly, eight of ten participants answering with *very probably* are *systems engineers*, *safety manager/engineers*, and *research engineers*. A *verification engineer* highlights the necessity of easing the semantics: “An engineer can make use of such tools only if it serves a user base. Formal methods offer great metrics to assert real-world processes, however it is limited today by its very semantic nature. One needs something on top to make this human understandable.” Further, 11 of 13 participants answering with *probably* are *systems engineers* and *safety manager/engineers*. A *system engineer* states that “it is necessary to integrate currently used/recommended methods. Changes from well known methods at software engineering team must be as easy as possible.”

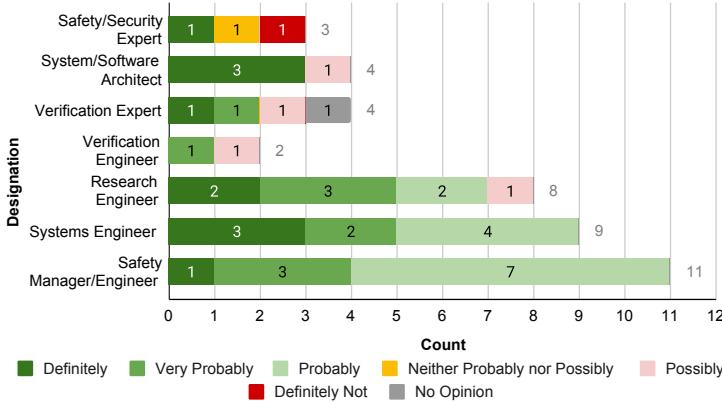


Figure 21: Results of whether formal methods are usable in real-world development processes grouped by the participants’ designations.

Apart from the positive opinions, a *safety/security expert* highlights that it “depends on what you consider formal methods. Type systems and similar methods are used heavily by compilers in the background already. Formal specifications in the form of LTL formulas for complex systems seem extremely challenging from a cost/benefit point of view”. All of the participants answering with *possibly* consider scalability as the major barrier. For example, a *verification expert* states that “scalability, time cost and, required technical knowledge constitute a barrier to usability.” Finally, the only participant answering with *definitely not* is a *safety/security expert* who states that “only a small amount of people will accept it, as too many people are not able to understand notations used.”

Summary. To sum up, 34 out of all 41 participants (83%) have positive opinions regarding the use of formal methods in real-world development processes (formal methods are *definitely*, *very probably*, and *probably* usable in such a context). According to the free-text comments, the main barriers for the application of formal methods in the real world are that they are not scalable, not suitable for all kinds of engineers, not easily understandable, and not in the state of “plug and play”, yet.

7 One-Group Pretest-Posttest Experiment (Part 2): Results and Analysis

The questionnaire listed in Table 3, gathered from 13 participants, corresponds to *Part 2* of our study, the one-group pretest-posttest experiment. The results of this experiment are summarized in the following.

7.1 Participants

We collected demographic information about the participants using the questions *DQ1* and *DQ2*.

7.1.1 Participants’ Experience in Formal Methods

The experience of participants in using formal methods is collected through question *DQ2* (Table 3). Possible answers correspond to scale *LS2* (Table 2). Likewise to *Q3* in Table 1 (*cf.* Section 6.1.1), the participants are asked to fill-in their experience in formal methods gained individually in academia and industry as well as their overall experience. Figure 22 shows the responses.

All of the 13 participants have gained experience in formal methods in academia, industry, or both. Particularly, all participants have academic experience. Six participants have <1 year and four participants have 4 to <6 years of experience gained in academia. Concerning industrial experience, four

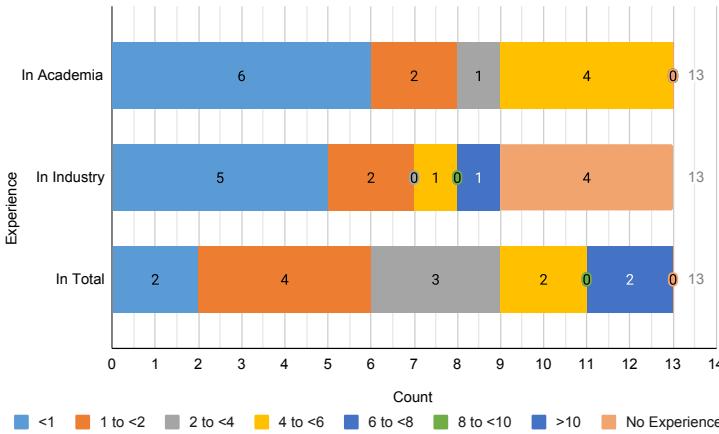


Figure 22: Experience of the participants in formal methods gained in academia, industry, and overall/in total.

participants have *no experience* and five participants have <1 year of experience. Finally, focusing on the overall experience, four participants have 1 to <2 years of experience, and three participants have 2 to <4 years of experience. Furthermore, two participants each have <1, 4 to <6, and >10 years of experience.

7.1.2 Participants' Knowledge in Formal Methods

Question *DQ1* (Table 3) asks the 13 participants to rate their knowledge in formal methods (answer according to scale *LS1* in Table 2). The responses of the participants are shown in Figure 23. Similar to the result of *Q2* in *Part 1* (*cf.* Section 6.1.2), all participants have rated their knowledge within the scale *novice* to *expert* while no answers are received for the scale *mastery* and *practical wisdom*. Thus, both *mastery* and *practical wisdom* scales will not be considered for the rest of the discussion. Among the 13 participants, three participants each rated themselves as *novices*, *advanced beginners*, *competent*, and *experts* while only one participant is rated as *proficient*. Figure 23 depicts the participants' knowledge together with their total experience in formal methods. All participants having <2 years of experience are rated as *novice* and *advanced beginners*, 2 to <4 years as *competent*, and 4 to <6 as well as > 10 years as *proficient* and *experts*.

7.2 Use Case for the Pretest

In this section, we discuss the use case for the pretest, a formally specified airbag system, that has been presented to the participants, as well as the collected responses for questions *TQ1* and *TQ2* (*cf.* Table 3).

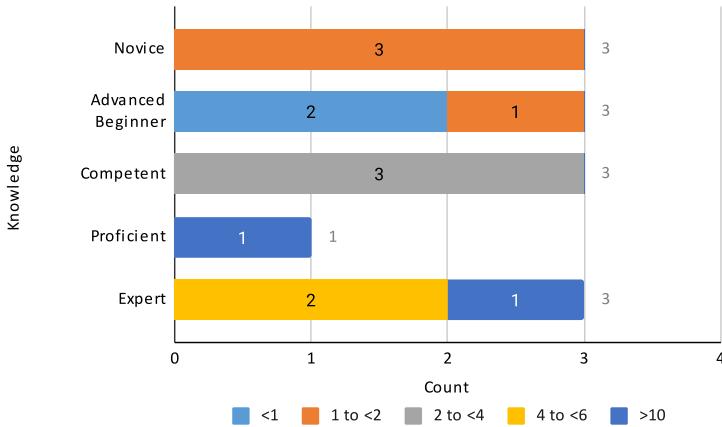


Figure 23: Participants’ knowledge in formal methods, along with their total experience in formal methods in terms of years.

7.2.1 Airbag System

The component model and specifications of the airbag system used in the pretest are shown in Figure 24. An explanation of the component model and their corresponding specifications were provided in-detail in a video to the participants. The airbag system consists of one parent component and two sub-components, *CollisionPlausibilization* and *AirbagController*. The behavior of the parent component is to activate the airbag system via the *exploded* signal whenever any of the sensor signals (*sen_front*, *sen_right*, *sen_left*, or *sen_back*) holds. To achieve this, the sub-component *CollisionPlausibilization* processes the sensor signals and provides the detection signal *collision_detected* as output. Finally, the sub-component *AirbagController* takes the signal *collision_detected* as input and provides the *exploded* signal to activate the actuator of the airbag system.

7.2.2 Difficulty of the Use Case and Understanding it

Questions *TQ1* and *TQ2* (Table 3) assess the participants’ difficulty and understanding of the airbag system use case. The participants’ responses are shown in Figure 25 (responses follow scale *LS5* in Table 2).

Difficulty of the use case. Among 13 participants, 11 participants perceive the airbag system use case as not difficult by answering within the scale *possibly*, *probably not*, or *definitely not*. Figure 26a shows responses for the difficulty of the airbag system use case based on the participants’ knowledge in formal methods (*cf.* Section 7.1.2).

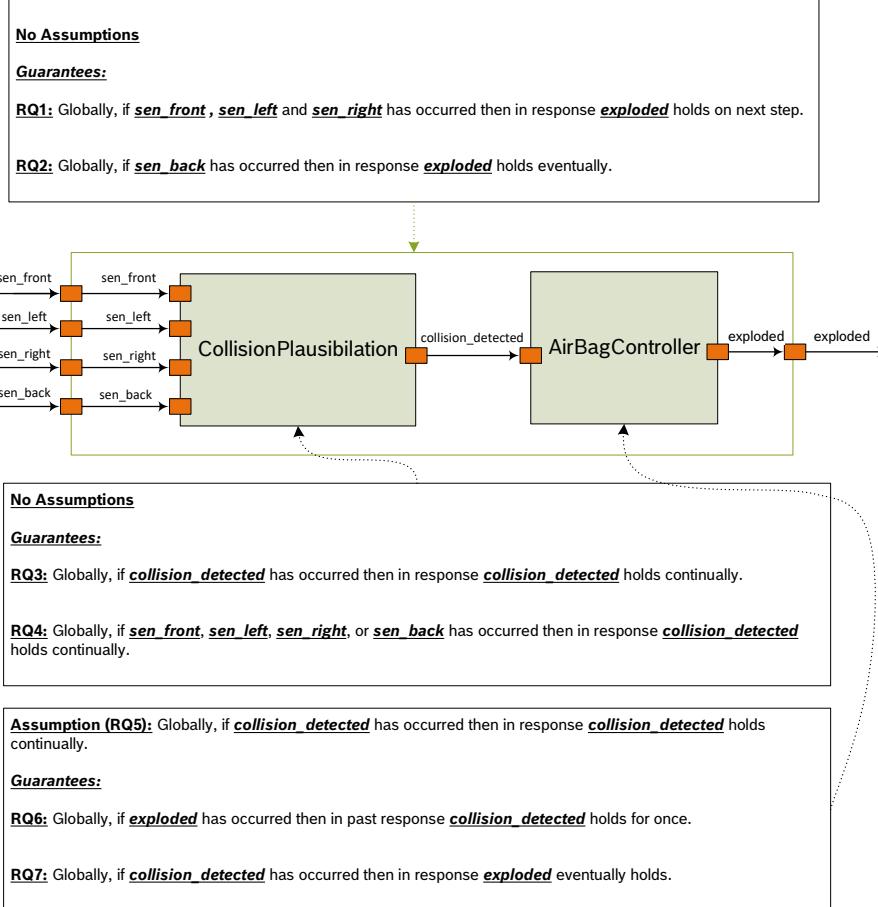


Figure 24: The component model and specifications of the airbag system.

Difficulty in understanding the use case. The responses to question *TQ2* is almost similar to those for *TQ1* (cf. Figure 25). A majority of participants (9 of 13) rate their understanding of the airbag system use case as not difficult by answering within the scale *possibly*, *probably not*, or *definitely not*. Figure 26b shows the responses for the difficulty of understanding the use case based on the participants' knowledge in formal methods. Among three participants as *advanced beginners*, one participant each answers *very probably* and *probably*. One participant each answering *very probably* and *definitely not* (two extreme ends) are an *advanced beginner* and a *competent* person in formal methods.

Summary. The answers show that most of the participants (11 of 13) perceive the airbag system as not being a difficult use case. Similarly, understanding the airbag system use case is also not rated as difficult (9 of 13).

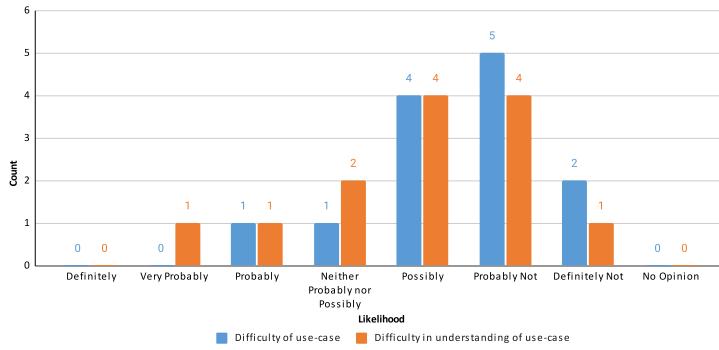


Figure 25: Difficulty of the airbag system use case and of understanding it.

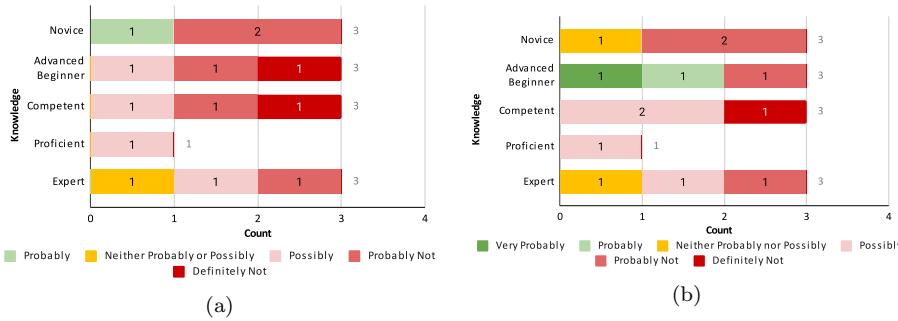


Figure 26: Difficulty of (a) the use case and (b) of understanding it grouped by the participants' knowledge in formal methods.

7.3 Use Case for the Posttest

For the posttest, we use a more complex use case, an electronic power steering system. As for the use case in the pretest, we ask the participants questions *TQ1* and *TQ2* (*cf.* Table 3).

7.3.1 Electronic Power Steering System

The Electronic Power Steering (EPS) (Bozzano et al. 2020) system is a Bosch product designed for highly-automated driving vehicles. It steers either based on input from the driver or commands from the vehicle bus. The ECU component of the EPS system, shown in Figure 27 and used for the posttest, has two redundant channels: a primary and a secondary channel. Each channel consists of three modes: master, slave, and passive. The nominal behavior is that one channel is master and the other one is slave. This synchronization is taken care of by the sub-component *interComDevice* in between the primary and secondary devices. As shown in Figure 27, if *torque_request_from_pd* does not hold and *torque_request_to_pd* holds, then the sub-component *pri-*

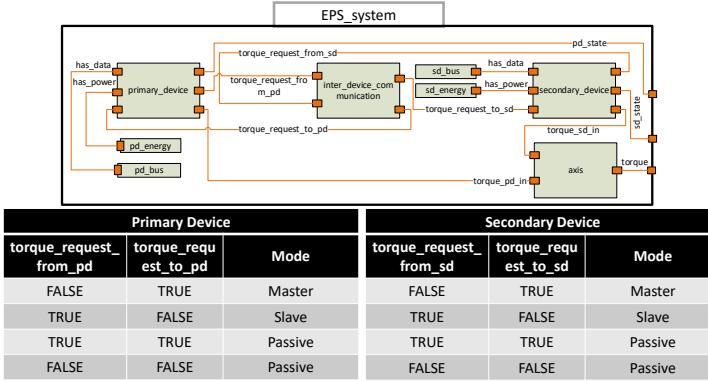


Figure 27: The component model and specifications of the electronic power steering (EPS) system.

mary_device is in master mode. Similarly, if *torque_request_from_sd* holds and *torque_request_to_pd* does not hold, then the sub-component *secondary_device* is in slave mode. In these two cases, *torque* is given to the system actuator.

7.3.2 Difficulty of the Use Case and Understanding it

As mentioned previously, the same questions *TQ1* and *TQ2* in Table 3 used in the pretest are again used in the posttest to collect the participants' opinions in assessing the difficulty of the electronic power steering system use case and of understanding it. The participants' responses are shown in Figure 28 (answers follow scale *LS5* in Table 2). To avoid any bias, we have used a less complex system for the pretest and more complex real-world project for the posttest. Thus, our early hypothesis is that a majority of participants perceives the use case and its understanding of the posttest as more complex. This is confirmed by the results shown in Figure 28. Overall, the motive of using the EPS use case for the posttest is to identify whether the proposed counterexample explanation approach is suitable for real-world systems with the use of formal methods.

Difficulty of the use case. Figure 28 shows that seven out of 13 participants perceive the EPS use case as difficult by answering with *very probably* or *probably*. One further participant answers with *neither probably nor possibly* and the remaining five participants perceive it as not difficult (*possibly*, *probably not*, and *definitely not*). Figure 29a groups the responses by the participants' knowledge in formal methods. Except for the knowledge level *competent*, all other knowledge levels predominantly perceive the use case as difficult.

Difficulty in understanding the use case. The results for *TQ2* on the difficulty of understanding the use case are similar to *TQ1*. Seven of 13 participants

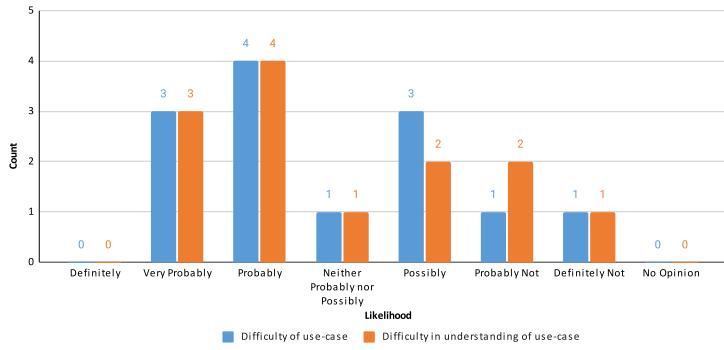


Figure 28: Difficulty of the EPS use case and of understanding it.

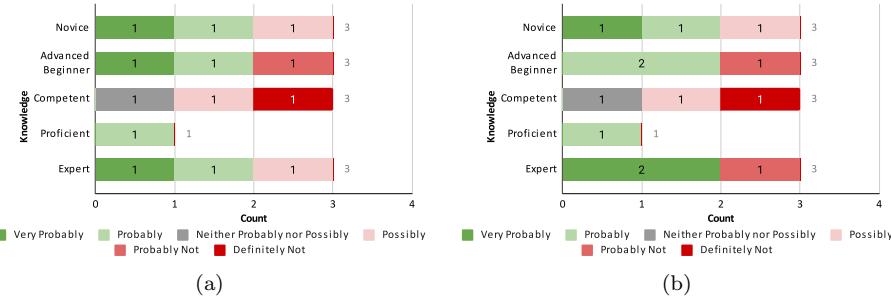


Figure 29: Difficulty of (a) the EPS use case and of (b) understanding it grouped by the participants' knowledge in formal methods.

perceive understanding the EPS system as difficult, one participant answers with *neither probably nor possibly*, and the remaining five participants perceive it as not difficult. The results are also similar to *TQ1* taking the participants' knowledge levels into account (Figure 29b).

Summary. The results for both the difficulty of the EPS use case and of understanding it are quite similar. The majority of participants (7 of 13) perceive both the EPS use case and understanding it as difficult.

7.4 Results of the Model Checker and Counterexample Explanation, and Understanding it

We use questions *TQ3* and *TQ4* (Table 3) to collect the participants' responses on understanding the model checker and counterexample explanation results shown during the pretest and protest to the participants. These results and responses are discussed in the following.

Violated Specifications

Note: *Highlighting is not provided by the model checker. We highlighted the inconsistent specification to ease the error comprehension of raw model checker results.*

```
TRUE & (TRUE -> G(CollisionPlausibilization.collision_detected -> G CollisionPlausibilization.collision_detected) &
G((CollisionPlausibilization.sen_front | CollisionPlausibilization.sen_right | CollisionPlausibilization.sen_left |
CollisionPlausibilization.sen_back) -> G CollisionPlausibilization.collision_detected) & (G(AirBagController.collision -> G
AirBagController.collision) -> (G(AirBagController.exploded -> O AirBagController.collision) & G(AirBagController.collision -> F
AirBagController.exploded)) & G(wire) -> G((sen_right & sen_left) & sen_front) -> X exploded) & G(sen_back -> F exploded)
```

Counterexample

-> State: 1.1 <- sen_front = TRUE sen_left = TRUE sen_right = TRUE sen_back = TRUE CollisionPlausibilization. sen_front = TRUE CollisionPlausibilization. sen_left = TRUE CollisionPlausibilization. sen_right = FALSE CollisionPlausibilization. sen_back = FALSE CollisionPlausibilization.collision_detected = TRUE AirBagController.collision_detected = TRUE AirBagController.exploded = TRUE exploded = TRUE wire = TRUE	-- Loop starts here -> State: 1.2 <- AirBagController.exploded = FALSE exploded = FALSE -> State: 1.3 <- AirBagController.exploded = TRUE exploded = TRUE -> State: 1.4 <- AirBagController.exploded = FALSE exploded = FALSE
--	--

Figure 30: Result of the model checker when verifying a refinement of the airbag system.

Model checker output. Figure 30 is the result generated by the model checker for the airbag system use case described in Section 7.2.1. This result is used during the pretest. If an inconsistency is identified by the model checker, the whole violated refinement specification and the counterexample to illustrate the erroneous behavior are shown to the participant. We have highlighted the inconsistent specifications in the whole violated refinement specification to avoid any bias. For example, the model checker output could be hard to interpret on first glance, which is not necessarily the case for our counterexample explanation approach since the relevant information is highlighted explicitly. This could mislead the participant at first glance in deciding the model checker output to be too difficult. Thus, to avoid this bias, highlighting the inconsistent specification in the model checker output would trigger the participant to understand and identify the inconsistency.

Counterexample explanation. The counterexample explanation shown in Figure 31 is used for the posttest. This explanation is the result of verifying a refinement and explaining a refinement inconsistency of the EPS use case (*cf.* Section 7.3.1). Instead of showing the complete violated refinement specification, the explanation presents the type of violation, list of inconsistent specifications, and their corresponding components. Further, instead of a concrete counterexample, highlighting of erroneous states and variables in the counterexample as well as explanations of the erroneous and expected nominal behavior are shown to ease the error comprehension for the participants.

Refinement Violation Explanation

Inconsistency in the decomposition of guarantee of parent-component EPS System. The guarantee “*Globally, it is always the case that pd_state = Master and sd_state = Slave holds*” of parent-component EPS System is inconsistent with the guarantee “*Globally, it is always the case that if pd_state = Master and torque_request_from_pd=FALSE and torque_request_to_pd = TRUE holds, then pd_state = Master and torque_request_from_pd = FALSE and torque_request_to_pd = FALSE holds in next transition*” of its sub-component primary_device.

Counterexample Explanation

The variable **EPS_system.pd_state** holds **Passive** in **State: 1.3**.

But **EPS_system.pd_state** is expected to be **Master** in **State: 1.3**.

```
-> State: 1.1 <
primary_device.pd_state = Master
secondary_device.sd_state = Slave
EPS_system.pd_state = Master
EPS_system.sd_state = Slave
primary_device.has_data = TRUE
primary_device.torque_request_from_pd = FALSE
primary_device.has_power = TRUE
primary_device.torque_pd_in = TRUE
primary_device.torque_request_to_pd = TRUE
secondary_device.has_data = TRUE
secondary_device.torque_request_from_sd = TRUE
secondary_device.has_power = TRUE
secondary_device.torque_sd_in = TRUE
secondary_device.torque_request_to_sd = FALSE
inter_device_communication.torque_request_from_sd=TRUE
inter_device_communication.torque_request_from_pd=FALSE
inter_device_communication.torque_request_to_sd = FALSE
inter_device_communication.torque_request_to_pd = TRUE
axis.torque_pd_in = TRUE
axis.torque_sd_in = TRUE
axis.torque = TRUE
EPS_system.torque = TRUE
wire = TRUE

-- Loop starts here
-> State: 1.2 <
primary_device.torque_request_to_pd = FALSE
inter_device_communication.torque_request_from_sd = FALSE

-> State: 1.3 < (Erroneous State)
primary_device.pd_state = Passive
EPS_system.pd_state = Passive (erroneous variable)
primary_device.torque_out = FALSE
primary_device.torque_request_to_pd = TRUE (erroneous variable)
inter_device_communication.torque_request_from_sd = TRUE
axis.torque_pd_in = FALSE

-> State: 1.4 <
primary_device.pd_state = Master
EPS_system.pd_state = Master
primary_device.torque_out = TRUE
primary_device.torque_request_to_pd = FALSE
inter_device_communication.torque_request_from_sd = FALSE
axis.torque_pd_in = TRUE
```

Figure 31: Result of the counterexample explanation approach when verifying a refinement of the EPS system.

Understanding the results. With questions *TQ3* and *TQ4*, we assess the understanding of the model checker output and counterexample explanation approach. Only a minority (6 of 13) of participants perceives the model checker output as easy to understand (these participants answer with *definitely*, *very probably*, and *probably*), while a clear majority participants (12 of 13) perceives understanding of the counterexample explanation approach as easy (Figure 32). Notably, no participant answers that understanding the results of the counterexample explanation approach is hard, while six of 13 participants still find the model checker output to be hard to understand (corresponding responses are *possibly* and *probably not*), even with the less complex use case (airbag system) and additional highlighting of the inconsistent specifications in the model checker output.

Understanding the results grouped by the participants' knowledge in formal methods. Figure 33a and Figure 33b show how the participants understand the model checker output and counterexample explanation based on their knowledge in formal methods. The understanding of the model checker output (Figure 33a) aligns with the participants' knowledge. For example, a majority of the participants answering that understanding is hard (*possibly* and *probably not*) are either *novice* or *advanced beginners*. On other hand, a majority of the

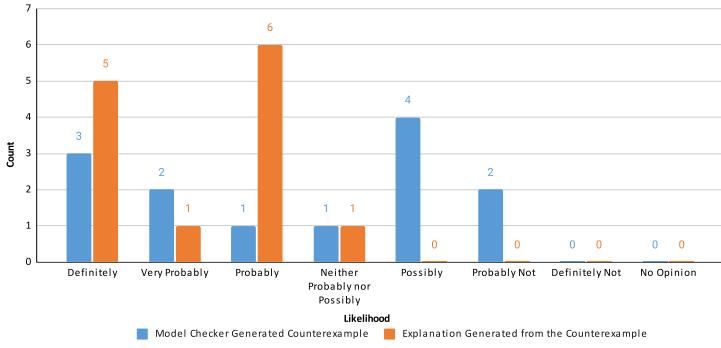


Figure 32: Results for understanding the results generated by the model checker and counterexample explanation approach.

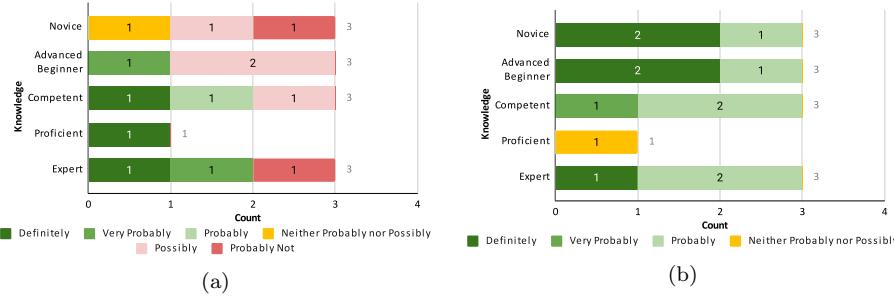


Figure 33: Responses for understanding the results generated (a) by the model checker and (b) by the counterexample explanation approach grouped by the participants' knowledge in formal methods.

participants answering that understanding is easy (*definitely*, *very probably*, and *probably*) belongs to the *competent*, *proficient*, or *expert* groups. However, as shown in Figure 33b, a majority of the participants belonging to the *novice* and *advanced beginner* groups perceive that understanding the counterexample explanation is *definitely* easy.

Summary. The raw output generated by the model checker contains the whole violated refinement specification and the counterexample to illustrate the erroneous behavior. The counterexample explanation approach, however, generates an explanation containing the type of violation, the inconsistent specification, and their corresponding components. Additionally, erroneous states and erroneous variables are highlighted in the counterexample. From the collected responses, 12 out of 13 participants perceive the understanding of the counterexample explanation result as easy, which contrasts the six out of the 13 participants who perceive the model checker output to be easy to understand.

7.5 Participants Responses for Task-Related Questions

Questions from $TQ5$ to $TQ9$ are task-related, in which participants answer based on their understanding of the model checker and counterexample explanation results. For these questions during the pretest and posttest, the participants should identify the inconsistent components ($TQ5$) and specifications ($TQ6$), the reason for the inconsistency ($TQ7$), a solution to fix the inconsistency ($TQ8$), and a nominal behavior of the system in terms of a correct state transition in place of the erroneous state transition ($TQ9$).

Identifying inconsistent components. Figure 34a shows the responses to $TQ5$, which requires from the participants to identify the inconsistent components based on the model checker result during the pretest and counterexample explanation during the posttest. Eight out of 13 participants identify both the inconsistent parent and sub-components during the pretest and ten participants during the posttest. Further three participants each identify *either* the sub-component or parent component correctly during both the pretest and posttest. Notably, no participant completely identifies the incorrect components based on the counterexample explanation.

Identifying inconsistent specifications. Figure 34b shows responses to $TQ6$, for which participants should identify the inconsistent specification from the model checker output and based on the counterexample explanation. Among the 13 participants, four participants have *no opinion* from the model checker output. From the remaining nine participants, four participants each identify the inconsistent specifications fully correct and completely incorrect, and one participant identifies parts of the inconsistent specification correctly. Concerning the responses of identifying the inconsistent specifications using the counterexample explanation, nine of 13 participants identify all inconsistent specifications correctly. The remaining four participants identify parts of specifications correctly.

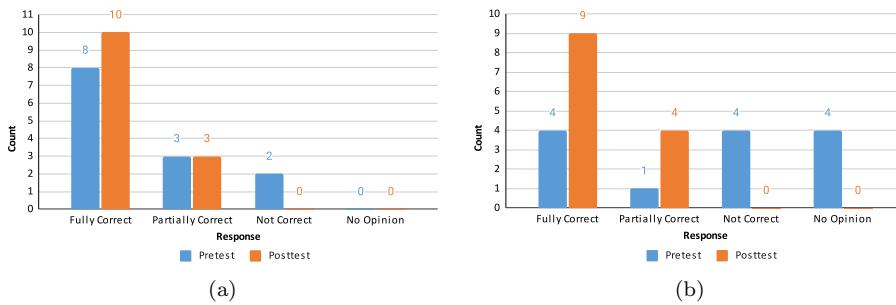


Figure 34: Results of identifying (a) inconsistent components and (b) inconsistent specifications from the model checker output (pretest) and counterexample explanation (posttest).

Among the five participants who identify the inconsistent specifications fully or partially based on the model checker output, the responses by four participants are fully correct in terms of the reason for the inconsistency (*TQ7*) and the appropriate fix (*TQ8*). On other hand, among 13 participants who identify the inconsistent specifications fully or partially using the counterexample explanation, the responses by 11 participants correctly identify the reason (*TQ7*) and by nine participants correctly identify the fix (*TQ8*).

Nominal system behavior in the counterexample. Among the 13 participants, six participants attempt to answer *TQ9* in the pretest and nine participants in the posttest to identify the expected system behavior in the counterexample, that is, to name a correct state transition in place of the erroneous state transition. Among the six participants in the pretest, four participants correctly name the expected behavior based on the model checker output, while seven of the nine participants in the posttest correctly name the expected behavior based on the counterexample explanation.

Summary. A majority of participants identifies both the inconsistent components and specifications correctly based on the counterexample explanation approach in the posttest, significantly more than based on the model checker output in the pretest. This shows that the counterexample explanation is well suited to identify inconsistent components and specifications, and that participants are able to explain the reason for and to fix the inconsistencies.

7.6 Participants' Opinion on Understanding the Model Checker Output and Counterexample Explanation

Questions *PRQ1* to *PRQ4* are used to collect the participants' opinions on understanding the model checker output during the pretest (*LS6* scale). Similarly, questions *POQ1* to *POQ4* are used to collect opinions on understanding the counterexample explanation during the posttest. Each group of four questions (*PRQ1* to *PRQ4* and *POQ1* to *POQ4*) mainly focus on four aspects: (1) better understanding, (2) quicker understanding, (3) confidence, and (4) added value. The responses for these four aspects are shown in Figures 35–38.

Better understanding. Figure 35 shows the results whether the participants think that the model checker output resp. the counterexample explanation helps understanding refinement inconsistencies. Nine out of 13 participants answer this question positive (*strongly agree*, *agree*, and *somewhat agree*) with respect to the model checker output in the pretest. On other hand, all of the 13 participants answer the question positive with respect to the counterexample explanation in the posttest. Out of these 13 participants, five participants each *strongly agree* and *agree*, proving that the counterexample explanation allows participants to better understand inconsistencies of refinements.

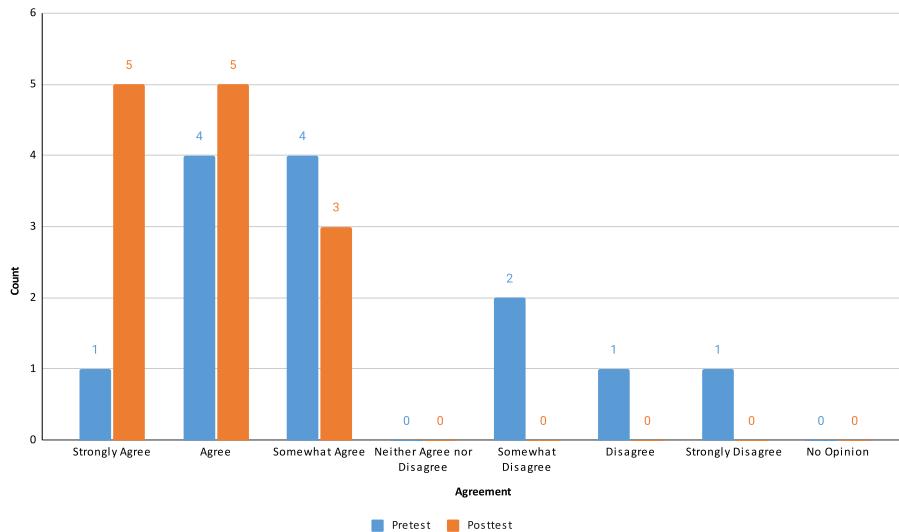


Figure 35: Better understanding.

Quicker understanding. Figure 36 shows the participants' responses regarding time saved based on the model checker output in the pretest resp. the counterexample explanation in the posttest. Regarding the model checker output (pretest), most participants *neither agree nor disagree* (5 participants). Among the remaining eight participants, six participants answer positively (*strongly*

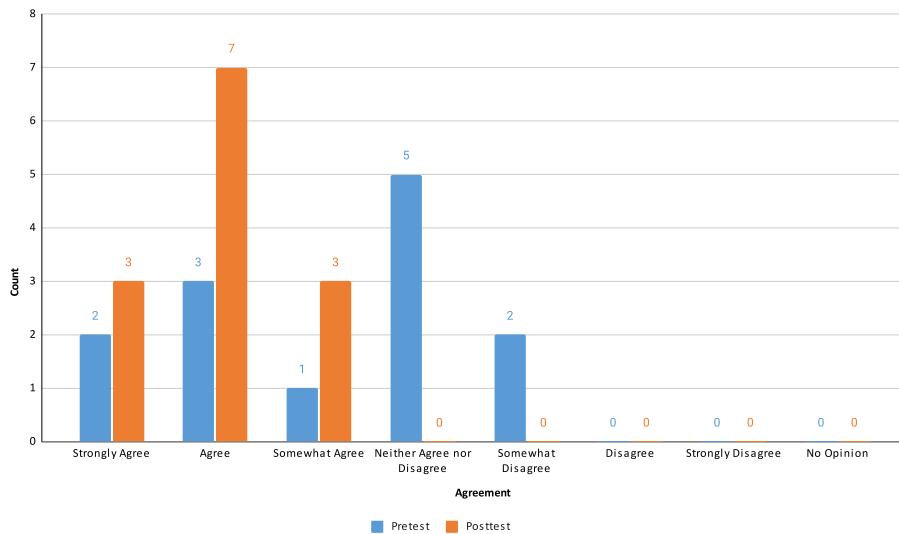


Figure 36: Quicker understanding.

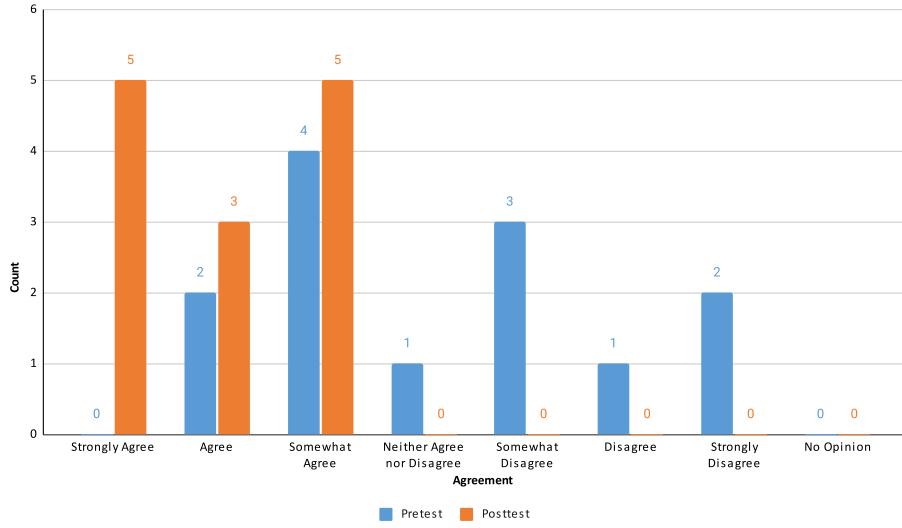


Figure 37: Confidence of understanding.

agree, agree, and somewhat agree) and two participants answer negatively (*somewhat disagree*).

Based on the counterexample explanation (posttest), a majority of the participants *agree* that this can save time (7 participants). From the remaining participants, three participants each answer with *strongly agree* and *somewhat agree*. No participants answers negatively, which strongly indicates that the provided counterexample explanation does indeed support a quicker understanding of inconsistencies.

Confidence of understanding. Figure 37 shows the participants' responses and depicts whether the output from the model checker or the counterexample explanation makes participants confident in their understanding of inconsistencies. An equal number of participants (6 participants each) answer either positively (*agree* and *somewhat disagree*) or negatively (*somewhat disagree*, *disagree*, and *strongly disagree*) for the pretest. For the posttest, however, all participants agree that the counterexample explanation helps to gain confidence with five participants responding *strongly agree*, five participants answering *somewhat agree*, and the remaining three participants responding *agree*.

Minimal added value. Figure 38 shows whether participants believe in an added value of the model checker output resp. the counterexample explanation by asking whether they think that the added value is minimal. Seven out of 13 participants answer with *disagree* for the statement that the model checker output provides only the minimal added value. Further, only one participant each answers *agree* and *somewhat agree*. For the counterexample explanation,

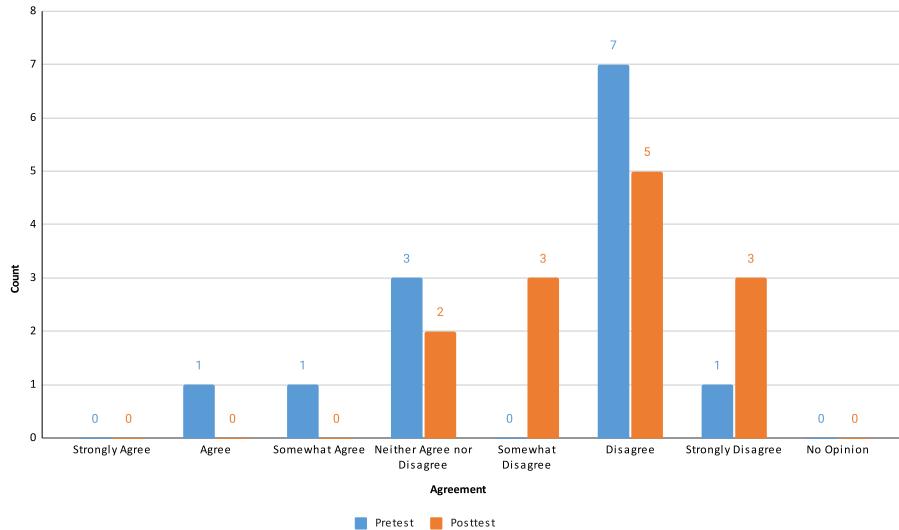


Figure 38: Minimal added value.

the largest share of participants (5 participants) *disagrees* that the explanation only adds minimal value to real-world projects. Further three participants each *somewhat disagree* and *strongly disagree*, two participants *neither agree nor disagree*. No participant agrees that the counterexample explanation does only provide a minimal added value.

Summary. Figures 35–38 show that the participants think that the counterexample approach helps in (1) better and (2) quicker understanding of inconsistencies, that it (3) raises their confidence in the analysis, and that (4) the provided value is not minimal. For all four aspects, answers were positive for the counterexample explanation in the posttest than for the model checker output in the pretest.

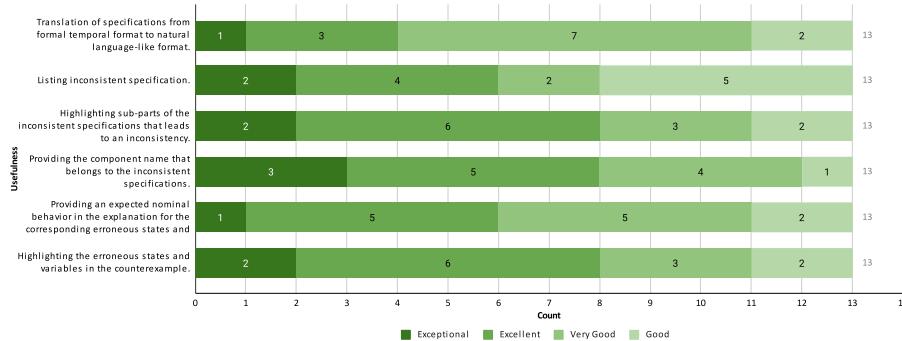


Figure 39: Participants' rating of counterexample explanation features.

7.7 Rating of Counterexample Explanation Features

Questions *FQ1* to *FQ6* are designed to collect the participants' opinions on the different features of the counterexample explanation approach (possible responses follow scale *LS7*). The responses are shown in Figure 39. Notably, all provided features are rated positively ranging from *good* to *exceptional*. No participant rated any feature negatively, that is, to be *fair*, *poor*, or *very poor*. In Figure 39, the first four features correspond to explaining the violated specification. The last two features correspond to explaining the counterexample.

The three features regarding highlighting the erroneous sub-parts in the inconsistent specification (feature #3 in Figure 39), listing the component name of the corresponding inconsistent specifications (feature #4), and highlighting the erroneous state in the counterexample (feature #6) were rated particularly helpful, each with eight participants answering either *exceptional* or *excellent*.

7.8 Feedback

Questions *FE1* to *FE8* are used to collect feedback from the participants on the counterexample explanation approach and using formal methods. Suggestions and responses are discussed in the following.

7.8.1 Comparison of Understanding Inconsistencies

With question *FE1*, we asked the participants for feedback according to scale *LS5* whether it is easier to understand the inconsistencies with the counterex-

Table 16: Participants' opinions on the ease of understanding inconsistencies with a counterexample explanation in contrast to a model checker output.

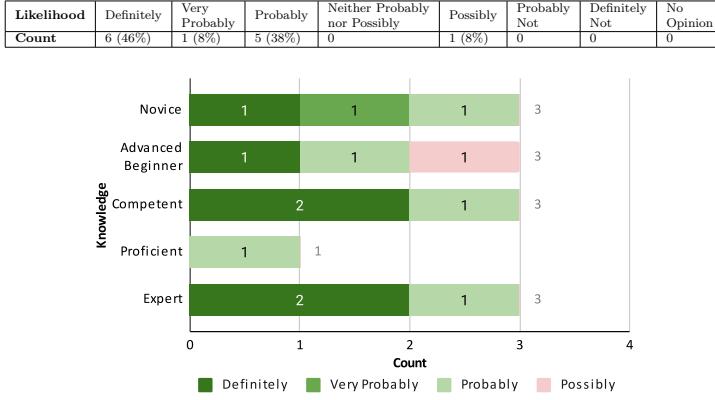


Figure 40: Participants' opinions on the ease of understanding inconsistencies with a counterexample explanation in contrast to a model checker output, grouped by the participants' knowledge in formal methods.

ample explanation approach than based on the original model checker output. From the 13 participants, the largest share with six participants answer with *definitely*, further five participants with *probably* (Table 16). Figure 40 shows the results grouped by the participants' knowledge in formal methods.

A majority of participants answer that understanding inconsistencies is easier with the counterexample explanation than with the original model checker output. Only one participant answers slightly hesitant (*possibly*).

7.8.2 Challenges in Analyzing Inconsistencies

With question *FE2*, we want to identify challenges that participants see in analyzing inconsistencies based on the counterexample explanation approach (answers as free-text comments). Nine out of 13 participants provided responses. Among them, four participants state that the main challenge in understanding stems from the complexity of the EPS system specifications and components. Further three participants state that the main challenge stems from not using formal methods frequently in their daily work. An example statement from a participant is the following: “*I think, the second example is much more complicated. As I did not work with formal specifications within the last 5 years, it was hard to get into the topic.*”. The remaining two participants highlight that it is still hard to fix the issues despite the counterexample explanation: “*Highlights and explanation do not provide a solution to the issue so the challenge to identify the root cause in the specification and removing it in a way to express the intended behavior still remains a challenge. But the approach helps to identify the root cause.*”

Table 17: Participants' opinions on the ease of maintaining refinement consistency during a refinement step with the counterexample explanation approach.



Figure 41: Participants' opinions on the ease of maintaining refinement consistency during a refinement step with the counterexample explanation approach, grouped by the participants' knowledge in formal methods.

7.8.3 Maintaining Refinement Consistency

With question *FE3*, we collect feedback according to scale *LS3* on whether it is easier to maintain the refinement consistency during a refinement step with the counterexample explanation approach. A majority of the participants answers positively (Table 17). Two participants answer with *extremely easy*, six with *easy*, four with *slightly easy*, and one with *neither hard nor easy*. Figure 41 shows the results based on participants' knowledge in formal methods.

7.8.4 Counterexample Explanation in Real-world Development

With question *FE4*, we collect feedback according to scale *LS5* from the participants on whether the proposed counterexample explanation approach is usable in the real-world development processes. Twelve participants provided responses. According to Table 18, a majority of participants (5) vote for *probably*, with ten participants in total answering positively (*definitely*, *very probably*, and *probably*). Figure 42 shows the participants' opinions on using the counterexample explanation approach in real-world development based on their knowledge in formal methods.

7.8.5 Counterexample Explanation with Formal Methods in Real-World Development

With question *FE5*, we collect feedback from participants according to answer scale *LS5* on whether engineers prefer to use the proposed counterexample

Table 18: Participants' opinions on using the counterexample explanation approach in real-world development.

Likelihood	Definitely	Very Probably	Probably	Neither Probably nor Possibly	Possibly	Probably Not	Definitely Not	No Opinion
Count	2 (15%)	3 (23%)	5 (38%)	0	2 (15%)	0	0	1 (8%)

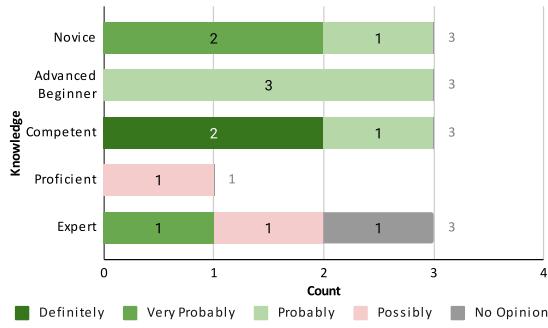


Figure 42: Participants' opinions on using the counterexample explanation approach in real-world development, grouped by the participants' knowledge in formal methods.

Table 19: Participants' opinions on using the counterexample explanation approach while using formal methods in real-world development.

Likelihood	Definitely	Very Probably	Probably	Neither Probably nor Possibly	Possibly	Probably Not	Definitely Not	No Opinion
Count	1 (8%)	1 (8%)	7 (54%)	0	3 (23%)	0	0	1 (8%)

explanation while using formal methods. Among 13 participants, 12 have responded (Table 19). Among these 12 participants, the largest share of seven participants answer with *probably*. One further participant each answers with *definitely* and *very probably*, the remaining three participants answer with *possibly*.

7.8.6 Counterexample Explanation in Your Project

With question *FE6*, we collect feedback from the participants according to answer scale *LS5* on whether engineers prefer to use the proposed counterexample explanation approach in their real-world projects. Four participants have *no opinion* and the remaining nine participants provided feedback (Table 20). Among these nine participants, five participants answer negatively (*possibly*, *probably not*, and *definitely not*), and four participants answer positively (*definitely*, *very probably*, and *probably*). These answers contrast the previously discussed feedback questions *FE1* to *FE5* where responses are more positive while for *FE6*, the majority of participants answer negatively. The reason for this negative rating as mentioned by the majority of participants are the currently used development processes and tools that do not fit well with formal methods or the counterexample explanation approach.

Table 20: Participants' opinions on using the counterexample explanation approach in the participants' projects.

Likelihood	Definitely	Very Probably	Probably	Neither Probably nor Possibly	Possibly	Probably Not	Definitely Not	No Opinion
Count	2 (15%)	0	2 (15%)	0	1 (8%)	3 (23%)	1 (8%)	4 (31%)

7.8.7 Further Improvements

The motive of questions *FE7* and *FE8* are to collect suggestions for further improvement of the counterexample explanation approach. Question *FE7* mainly focuses on collecting participants' feedback on whether providing a list of possible solutions/fixes would be helpful for engineers. This feedback is collected according to answer scale *LS5*. *FE8* is an open question that allows participants to suggest further improvements as free-text comments. The responses to *FE7* are shown in Table 21. Most of the answers are positive, meaning that providing a list of possible solutions/fixes would be helpful to participants.

Table 21: Participants' opinions on the usefulness of providing a list of possible solutions/fixes to a refinement inconsistency.

Likelihood	Definitely	Very Probably	Probably	Neither Probably nor Possibly	Possibly	Probably Not	Definitely Not	No Opinion
Count	5 (38%)	4 (31%)	2 (15%)	1 (8%)	1 (8%)	0	0	0

With question *FE8*, we collected general suggestions from the participants. We received eight responses that suggest improvement for using the counterexample explanation approach. A majority suggests to provide training to use formal methods, and improve or develop the visualization of the explanation by associating and integrating it with the tools used within Bosch, *e.g.*, IBM Rhapsody and DOORS Next Generation (DNG).

8 Discussion

In this section, we discuss the findings of the two user study phases following the research questions and points to be investigated (*cf.* Section 5.1).

8.1 RQ1 – Challenges in Identifying Inconsistent Specifications

Research question *RQ1* gathers challenges in identifying inconsistent formal specifications that are introduced during the refinement of a system.

Understanding formal notations is difficult for engineers. Formal methods may play a crucial role on the left-hand side of the V-model (Weber 2009), where *systems engineers* and *safety manager/engineers* are mainly involved, to avoid major flaws in early design decisions. In our *user survey* Phase 1, 20 of 41 participants are *system engineers* and *safety managers/engineers* (Section 6.1.3). The majority of them perceives understanding of formal notations as hard to some degree (*cf.* Section 6.2). Additionally, we noticed that the complexity of understanding depends on years of experience in using formal methods. Results further show that introducing formal methods to engineering teams with little experience in formal methods is challenging.

Identifying inconsistent specifications is difficult for engineers. Results in Section 6.3 clearly show that a majority of Phase 1 participants perceive the identification and understanding of inconsistent specifications as *hard* to some degree and that it consumes significant time. Results are similar for maintaining and verifying the refinement consistency (Section 6.4).

Qualitative statements by the participants note that identifying and understanding of inconsistent specifications highly depends on the size of the system and the number of its requirements. These statements emphasize the question whether the usage of formal methods for industrial system and at

industrial scale is possible. For example, the automobile sector is now developing systems that are quickly expanding in size and complexity as a result of highly automated driving. System and requirements are not only complex but also frequently changing, for instance, due to security demands.

Our initial expectation was that more of our participants perform *manual inspections/reviews* rather than using automated tools like *model checker*, *simulators*, and *reasoners* in their development projects. Thus, our hypothesis is that a majority of participants will answer that identifying inconsistent specifications and understanding is very complex. However, on the contrary, a majority of participants use *model checkers* (Section 6.3.5). Even though a majority uses automated tools like *model checkers*, a majority still answers that the identification and understanding of inconsistent specifications are *hard*. A reason for this might be the complexity of verification tools and their generated results, raising the need to make them more user-friendly to be used by engineers without having in-depth knowledge of formal methods.

8.2 RQ2 – Benefit of Formal Methods to Development Processes

RQ2 gathers insights on whether the identification of inconsistent specifications and usage of formal methods are beneficial to real-world development processes.

Using formal methods can make the system safer. Functionalities of automotive systems increase expeditiously, resulting in more (safety) requirements to avoid any unintended behavior. Thus, performing safety analysis early on the left-hand side of the V-model (Weber 2009) is crucial to help reducing the number of errors identified later during the validation. A majority of participants agrees that formal verification can make the system safer and be a benefit to the functional safety (*cf.* Section 6.5 and Section 6.6). Although a majority of the participants have a positive opinion on using formal verification, based on the qualitative answers from the participants, there is a discussion whether formal verification is usable and scalable to real-world systems. Specifically, a majority participants indicate that the usage of formal methods could be improved by making formal notations easier to understand (*cf.* Section 6.6.2).

Identifying inconsistent specifications is beneficial in real-world development processes. Eliciting requirements, refining requirements, and developing system architectures are the initial steps in the V-model (Weber 2009). Thus, requirement elicitation and refinement of those requirements are crucial as they serve as a basis for further system development and safety analysis. Errors and inconsistencies introduced in these early phases, identified only in later development stages, become costly and may lead to catastrophic events. There is a high possibility that most safety-critical errors are identified late during the validation phase in industry (Pohl and Rupp 2011). Therefore, to identify errors in the requirements during the initial stages performing manual reviews

(*e.g.*, inspections) does not seem to be sufficient or an efficient approach. This motivates the usage of automated methods like formal verification and simulation to help identifying errors in requirements and overcoming challenges of manual reviews. Although a majority of participants answers that using and understanding formal verification are complex, the majority agrees that identifying inconsistent formal specifications is beneficial for safety analysis and makes a system safer (*cf.* Section 6.5.3).

8.3 RQ3 – Easing the Use of Formal Methods

Insights for *RQ3* are drawn from Phase 2, the one-group pretest-posttest experiment. *RQ3* gathers insights whether engineers prefer to use formal methods (model checkers particularly) if the difficulty for understanding verification results to identify inconsistent specifications is reduced, in particular with the counterexample explanation approach.

The counterexample explanation approach eases the comprehension when compared to the interpretation of the raw model checker output. Six of 13 participants of the one-group pretest-posttest experiment answer that they understand the verification result generated by the model checker for the airbag system (Section 7.4). Our initial hypothesis was that by proving an additional user-friendly counterexample explanation, it would be easier for engineers to understand the error as well as that it can ease the usage of formal methods among engineers. Indications for this are already drawn from *Part 1*, where a majority answers that making formal notations easier to understand can improve the usage of formal methods (Section 6.6.2). Finally, the results discussed in Section 7.4 support this hypothesis, as a stark majority of 12 of 13 participants prefer the counterexample explanation compared to understanding the model checker output. In summary, improving the usability and understandability aspects of formal notations can promote the use of formal methods.

Additionally, the results collected for the four aspects of a better understanding, quicker understanding, confidence of understanding, and added value (*cf.* Section 7.6) validate this hypothesis, as the counterexample explanation improves all four aspects compared to the raw model checker output presented otherwise to engineers.

It is possible for engineers to identify and fix inconsistent specifications based on the counterexample explanation approach. Apart from collecting the participants' opinions, we can also rely on the experiment, which let participants perform tasks for the provided use cases. The results presented in Section 7.8.1 evaluate the difference of correct and incorrect answers in finding inconsistent components between pretest and posttest and shows only a minor difference between working with raw results and the counterexample explanation.

However, this is not the case for the identification of inconsistent specifications. With the raw model checker output, only five of 13 participants identify

either fully or partially correct the inconsistent specifications. While with the counterexample explanation, nine of 13 participants were able to identify the complete set of inconsistent specifications and also correctly explained the reason of the inconsistency by understanding the explanation. This strongly shows that a counterexample explanation can indeed improve the error comprehension and providing such an explanation can promote the use of formal methods among engineers.

The counterexample explanation approach can promote formal verification and usage of model checking in real-world development processes. From the collected responses, a majority of participants have a positive opinion as a counterexample explanation could support maintaining refinement consistency, could be usable in real-world development process, and could be used while using formal methods (*cf.* Section 7.8). These results clearly indicate that a counterexample explanation approach could be one possible way along with other possible options like property specification patterns to improve the usability aspects of formal methods. However, for the question whether the participants are interested to use the counterexample explanation approach in their project, the response is contradictory where only a minimal number of participants are interested to use it. A major challenge mentioned by the participants is that tools currently used in their projects do not support integrating the proposed counterexample explanation approach. This shows that integrating the existing verification tools with industrial tools needs to be one of the prime focus to improve the usage of formal methods. Nevertheless, such an integration cannot be achieved easily since larger organizations such as Bosch typically use different tools for different projects. Thus, focusing on adaption and integration of each tool individually is not a trivial task.

9 Threats to Validity

In this section, we discuss threats that might jeopardize the validity of our study results as well as on measures we take to reduce these threats. We consider threats to validity as discussed by Wohlin et al. (2012), Kitchenham and Pfleeger (2008), and Campbell and Stanley (1963). In the following, we structure them according to construct, internal, and external validity.

Construct validity. The prime threats to construct validity are related to the completeness of the questionnaire and in phrasing questions in a way that is understood by all participants in the same way. To mitigate these threats, we have considered the following steps in our research method: (i) we incorporated feedback from two senior engineers having background in formal methods and model checking, (ii) we incorporated feedback regarding unbiased questions from a psychologist, and (iii) we performed a pilot test with five research engineers to check for completeness and understandability.

Internal validity. The critical internal threat to be considered for the *user survey* is the selection of participants. Since we followed snowball sampling for the participant selection, there could be a possibility of several participants working in the same project, which could bias the final result. Therefore, we considered at most first four participants from each project and neglected further project members.

We consider threats to internal validity listed by Campbell and Stanley (1963) for the pretest-posttest experiment. To mitigate the *history* and *maturity* threats, we performed the posttest experiment within fifteen days following the pretest experiment. The most severe threats to be considered in this experimental design are *testing* and *instrumentation*. Those threats arise because participants get overwhelmed with the intervention including the fact that we have developed the counterexample explanation approach. Consequently, participants could answer more positively in the posttest experiment than the actual value due to the intervention. To mitigate these threats, participants conduct the study anonymously and we explicitly emphasized to the participants that the obtained study results would serve as a reference in the future to use our counterexample explanation approach for real-world projects at Bosch. Additionally, to avoid overwhelmed responses and accept only valid responses, we have added the task questions *TQ1–TQ9* (Table 3); And the response is accepted as valid only if the participant attempted to answer at least some part of these questions. Further, to reduce biasing between the pretest and posttest experiment, the use case of an airbag system (a toy example) used in the pretest is significantly less complex than the use case of the Bosch EPS system used in the posttest. However, to adjust the difficulty level of the systems used for the experiment, we used feedback from the pilot study with five research engineers. Basically, adjustment of difficulty is done by increasing or decreasing the number of components and size of the specifications which have to be understood by the participants.

Finally, another internal threat is to present the model checker's raw output with the inconsistent specification highlighted by us to the participants in the pretest. This could bias the participants' opinions that the model checker's output included the highlighted parts is easier to interpret than it actually is in practice where the highlighted parts are not available. As such, we can rather expect larger benefits of our counterexample explanation approach in practice than we observed it in the one-group pretest-posttest experiment.

External validity. To avoid polluting results, we do not force the participants to select an option from an answer scale for every question. For example, the participants could choose the option *No Opinion*, which supports in achieving actual results. However, the participants have the choice to enter a reason as a qualitative statement if they do not want to select any option. One of the severe drawbacks of the one-group pretest-posttest experiment is its generalization. However, the benefit of our study is that we used a real-world EPS system for the posttest experiment, and the participants are professional engineers who work on real-world automotive projects at Bosch.

10 Conclusion

Our user study was designed to i) identify the motivation, challenges, and applicability of formal methods in industry and ii) evaluate if the proposed counterexample explanation approach matches the identified challenges.

To identify the motivation, challenges, and applicability of formal methods in industry, we conducted an extensive survey with 41 participants of various business units and disciplines within Bosch as a first phase of the user study. Responses show that the majority of the participants is positive regarding the use of formal methods in real-world development processes. However, participants identify that incomplete formal models in industry, understanding formal notations, as well as understanding verification results, e.g., produced by a model checker, still remain a challenge for adoption of formal methods in practice. Identifying refinement inconsistencies gets more complex with the system getting more complex and the number of requirements increasing. Apart from understanding and scalability challenges, one of the major challenge in using verification tools is a lack of training for engineers.

As a second part of the user study, we performed a one group pretest-posttest study with 13 participants of various Bosch business units to evaluate if the proposed counterexample explanation approach is capable of supporting the use of formal methods in industry. Results from the experiments as well as collected opinions from participants prove that the approach helps in i) better understanding and ii) quicker understanding inconsistencies, that it iii) raises their confidence in the analysis, and that iv) it provide value for the development of safety-critical projects in industry.

As researchers and educators in formal methods, we should strive to make our notations and tools accessible to non-experts. – Clarke and Wing (1996)

Future directions. To leverage formal methods in real-world development processes, one of the most suitable means is to provide education and training in formal methods. On the one hand, universities can teach the foundations of formal methods to students (*e.g.*, temporal logics). On the other hand, companies can teach the skills required in the specific industrial context (*e.g.*, considering the domain and tooling) to people entering industry as well as upskill existing employees to understand the foundations of formal methods. By providing education structured training in formal methods either in universities or companies, hesitancy in using formal methods could be reduced and the benefits of formal methods could be reaped.

Looking at the results of evaluating our counterexample explanation approach, it is clear that understanding of verification results is easier with a counterexample explanation than with the direct output of model checker. In future, similar explanations need to be generated for different model checkers, domain-specific system models and requirements, as well as integrated with project-specific tool chains. Furthermore, instead of only providing explanations that illustrate the error, providing suggestion to fix those errors could

help to improve the agility to perform verification iteratively and thus, to support round-trip engineering.

Acknowledgements We would like to express our gratitude to all engineers who participated in our study for their time spent and their valuable responses. Furthermore, we would like to thank Amalinda Post, Igor Menzel, and Kevin Heiner for their support in designing the study and improving the questionnaire.

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

Data Availability The data collected during the studies is not openly available due to reasons of sensitivity to Bosch and the privacy of the engineers. The data can be made available from the corresponding author upon reasonable request.

References

- J. Abrial. Formal methods in industry: achievements, problems, future. In L. J. Osterweil, H. D. Rombach, and M. L. Soffa, editors, *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, pages 761–768. ACM, 2006. doi: 10.1145/1134285.1134406.
- M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, and A. Tang. Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. *IEEE Trans. Software Eng.*, 41(7):620–638, 2015. doi: 10.1109/TSE.2015.2398877.
- E. R. Babbie. *The basics of social research*. Cengage learning, 2016.
- C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.
- G. Barbon, V. Leroy, and G. Salaün. Debugging of behavioural models with CLEAR. In T. Vojnar and L. Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 386–392. Springer, 2019. doi: 10.1007/978-3-030-17462-0_26.
- J. Bicarregui, J. S. Fitzgerald, P. G. Larsen, and J. C. P. Woodcock. Industrial practice in formal methods: A review. In A. Cavalcanti and D. Dams, editors, *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*, pages 810–813. Springer, 2009. doi: 10.1007/978-3-642-05089-3_52.

- J. P. Bowen and P. T. Breuer. Formal methods communities of practice: A survey of personal experience. In A. Cerone, M. Autili, A. Bucaioni, C. Gomes, P. Graziani, M. Palmieri, M. Temperini, and G. Venture, editors, *Software Engineering and Formal Methods. SEFM 2021 Collocated Workshops - CIFMA, CoSim-CPS, OpenCERT, ASYDE, Virtual Event, December 6-10, 2021, Revised Selected Papers*, volume 13230 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2021. doi: 10.1007/978-3-031-12429-7_21.
- M. Bozzano, P. Munk, M. Schweizer, S. Tonetta, and V. Vozárová. Model-based safety analysis of mode transitions. In A. Casimiro, F. Ortmeyer, F. Bitsch, and P. Ferreira, editors, *Computer Safety, Reliability, and Security - 39th International Conference, SAFECOMP 2020, Lisbon, Portugal, September 16-18, 2020, Proceedings*, volume 12234 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2020. doi: 10.1007/978-3-030-54549-9_7.
- D. T. Campbell and J. C. Stanley. *Experimental and quasi-experimental designs for research*. Rand McNally Chicago, 1963.
- A. Cimatti and S. Tonetta. A property-based proof system for contract-based design. In *38th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2012, Cesme, Izmir, Turkey, September 5-8, 2012*, pages 21–28, 2012.
- A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A new symbolic model checker. *Int. J. Softw. Tools Technol. Transf.*, 2(4):410–425, 2000. doi: 10.1007/s100090050046.
- E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, 1996. doi: 10.1145/242223.242257.
- E. M. Clarke, O. Grumberg, D. Kroening, D. A. Peled, and H. Veith. *Model checking, 2nd Edition*. MIT Press, 2018a. ISBN 978-0-262-03883-6.
- E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018b. ISBN 978-3-319-10574-1. doi: 10.1007/978-3-319-10575-8.
- J. A. Davis, M. A. Clark, D. D. Cofer, A. Fifarek, J. Hinchman, J. A. Hoffman, B. W. Hulbert, S. P. Miller, and L. G. Wagner. Study on the barriers to the industrial adoption of formal methods. In C. Pecheur and M. Dierkes, editors, *Formal Methods for Industrial Critical Systems - 18th International Workshop, FMICS 2013, Madrid, Spain, September 23-24, 2013. Proceedings*, volume 8187 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2013. doi: 10.1007/978-3-642-41010-9_5.
- L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24.

- M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In B. W. Boehm, D. Garlan, and J. Kramer, editors, *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999*, pages 411–420. ACM, 1999. doi: 10.1145/302405.302672.
- A. Ferrari and M. H. ter Beek. Formal methods in railways: A systematic mapping study. *ACM Comput. Surv.*, 55(4):69:1–69:37, 2023. doi: 10.1145/3520480.
- A. Ferrari, M. H. ter Beek, F. Mazzanti, D. Basile, A. Fantechi, S. Gnesi, A. Piattino, and D. Trentini. Survey on formal methods and tools in railways: The astrail approach. In S. C. Dutilleul, T. Lecomte, and A. B. Romanovsky, editors, *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - Third International Conference, RSSRail 2019, Lille, France, June 4-6, 2019, Proceedings*, volume 11495 of *Lecture Notes in Computer Science*, pages 226–241. Springer, 2019. doi: 10.1007/978-3-030-18744-6_15.
- A. Fink. *The survey handbook*. sage, 2003.
- S. Friedenthal, A. Moore, and R. Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.
- H. Garavel, M. H. ter Beek, and J. van de Pol. The 2020 expert survey on formal methods. In *Formal Methods for Industrial Critical Systems - 25th International Conference, FMICS 2020, Vienna, Austria, September 2-3, 2020, Proceedings*, pages 3–69, 2020. doi: 10.1007/978-3-030-58298-2_1.
- C. Gerking, W. Schäfer, S. Dziwok, and C. Heinemann. Domain-specific model checking for cyber-physical systems. In M. Famelis, D. Ratiu, M. Seidl, and G. M. K. Selim, editors, *Proceedings of the 12th Workshop on Model-Driven Engineering, Verification and Validation co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, MoDeVVa@MoDELS 2015, Ottawa, Canada, September 29, 2015*, volume 1514 of *CEUR Workshop Proceedings*, pages 18–27. CEUR-WS.org, 2015. URL <http://ceur-ws.org/Vol-1514/paper3.pdf>.
- D. Giannakopoulou, T. Pressburger, A. Mavridou, and J. Schumann. Generation of formal requirements from structured natural language. In N. H. Madhavji, L. Pasquale, A. Ferrari, and S. Gnesi, editors, *Requirements Engineering: Foundation for Software Quality - 26th International Working Conference, REFSQ 2020, Pisa, Italy, March 24-27, 2020, Proceedings [REFSQ 2020 was postponed]*, volume 12045 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2020. doi: 10.1007/978-3-030-44429-7_2.
- C. Gladisch, T. Heinz, C. Heinemann, J. Oehlerking, A. von Vietinghoff, and T. Pfitzer. Experience paper: Search-based testing in automated driving control applications. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 26–37, 2019. doi: 10.1109/ASE.2019.00013.
- M. Gleirscher and D. Marmsober. Formal methods in dependable systems engineering: a survey of professionals from europe and north america. *Empir. Softw. Eng.*, 25(6):4473–4546, 2020. doi: 10.1007/s10664-020-09836-5.

- O. Grumberg and H. Veith, editors. *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*, 2008. Springer. ISBN 978-3-540-69849-4. doi: 10.1007/978-3-540-69850-0.
- L. Grunske. Specification patterns for probabilistic quality properties. In W. Schäfer, M. B. Dwyer, and V. Gruhn, editors, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 31–40. ACM, 2008. doi: 10.1145/1368088.1368094.
- C. L. Heitmeyer. On the need for practical formal methods. In A. P. Ravn and H. Rischel, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 5th International Symposium, FTRTFT'98, Lyngby, Denmark, September 14-18, 1998, Proceedings*, volume 1486 of *Lecture Notes in Computer Science*, pages 18–26. Springer, 1998. doi: 10.1007/BFb0055332.
- C. B. Jones and M. Thomas. The development and deployment of formal methods in the UK. *Formal Aspects Comput.*, 34(1):1–21, 2022. doi: 10.1145/3522577.
- B. Kaiser, R. Weber, M. Oertel, E. Böde, B. M. Nejad, and J. Zander. Contract-based design of embedded systems integrating nominal behavior and safety. *Complex Syst. Informatics Model. Q.*, 4:66–91, 2015. doi: 10.7250/csimg.2015-4.05.
- A. P. Kaleeswaran, A. Nordmann, T. Vogel, and L. Grunske. Counterexample interpretation for contract-based design. In *Model-Based Safety and Assessment - 7th International Symposium, IMBSA 2020, Lisbon, Portugal, September 14-16, 2020, Proceedings*, pages 99–114, 2020.
- A. P. Kaleeswaran, A. Nordmann, T. Vogel, and L. Grunske. A user-study protocol for evaluation of formal verification results and their explanation. *CoRR*, abs/2108.06376, 2021. URL <https://arxiv.org/abs/2108.06376>.
- A. P. Kaleeswaran, A. Nordmann, T. Vogel, and L. Grunske. A systematic literature review on counterexample explanation. *Information and Software Technology*, 145:106800, 2022. ISSN 0950-5849. doi: <https://doi.org/10.1016/j.infsof.2021.106800>.
- M. Khazeev, H. Aslam, D. de Carvalho, M. Mazzara, J. Bruel, and J. A. Brown. Reflections on teaching formal methods for software development in higher education. In J. Bruel, A. Capozucca, M. Mazzara, B. Meyer, A. Naumchev, and A. Sadovskykh, editors, *Frontiers in Software Engineering Education - First International Workshop, FISEE 2019, Villebrumier, France, November 11-13, 2019, Invited Papers*, volume 12271 of *Lecture Notes in Computer Science*, pages 28–41. Springer, 2019. doi: 10.1007/978-3-030-57663-9_3.
- B. A. Kitchenham and S. L. Pfleeger. Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92. 2008. doi: 10.1007/978-1-84800-044-5_3.
- S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 372–381, 2005. doi: 10.1145/1062455.1062526.

- F. Kossak, A. Mashkoor, V. Geist, and C. Illibauer. Improving the understandability of formal specifications: An experience report. In C. Salinesi and I. van de Weerd, editors, *Requirements Engineering: Foundation for Software Quality - 20th International Working Conference, REFSQ 2014, Essen, Germany, April 7-10, 2014. Proceedings*, volume 8396 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2014. doi: 10.1007/978-3-319-05843-6_14.
- K. L. McMillan. The smv language. *Cadence Berkeley Labs*, pages 1–49, 1999.
- W. L. Neuman. *Basics of social research*. Pearson/Allyn and Bacon, 2014.
- L. C. Paulson. *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994. ISBN 3-540-58244-4. doi: 10.1007/BFb0030541.
- A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi: 10.1109/SFCS.1977.32.
- K. Pohl and C. Rupp. *Requirements Engineering Fundamentals - A Study Guide for the Certified Professional for Requirements Engineering Exam: Foundation Level - IREB compliant*. rockynook, 2011. ISBN 978-1-933952-81-9.
- A. Post and J. Hoenicke. Formalization and analysis of real-time requirements: A feasibility study at BOSCH. In *Verified Software: Theories, Tools, Experiments - 4th International Conference, VSTTE 2012, Philadelphia, PA, USA, January 28-29, 2012. Proceedings*, pages 225–240, 2012.
- A. Post, I. Menzel, J. Hoenicke, and A. Podelski. Automotive behavioral requirements expressed in a specification pattern system: a case study at BOSCH. *Requir. Eng.*, 17(1):19–33, 2012.
- D. Ratiu, A. Nordmann, P. Munk, C. Carlan, and M. Voelter. *FASTEN: An Extensible Platform to Experiment with Rigorous Modeling of Safety-Critical Systems*, pages 131–164. Springer International Publishing, 2021. ISBN 978-3-030-73758-0. doi: 10.1007/978-3-030-73758-0_5.
- A. Reid, L. Church, S. Flur, S. de Haas, M. Johnson, and B. Laurie. Towards making formal methods normal: meeting developers where they are. *CoRR*, abs/2010.16345, 2020. URL <https://arxiv.org/abs/2010.16345>.
- N. B. Robbins and R. M. Heiberger. Plotting likert and other rating scales. In *Proceedings of the 2011 Joint Statistical Meeting*, volume 1, 2011.
- C. Robson and K. McCartan. *Real world research*. John Wiley & Sons, 2016.
- P. Rodrigues, M. Ecar, S. V. Menezes, J. P. S. da Silva, G. T. A. Guedes, and E. M. Rodrigues. Empirical evaluation of formal method for requirements specification in agile approaches. In C. Boscaroli, C. A. Costa, S. de Avila e Silva, and D. L. Notari, editors, *Proceedings of the XIV Brazilian Symposium on Information Systems, SBSI 2018, Caxias do Sul, Brazil, June 04-08, 2018*, pages 53:1–53:8. ACM, 2018. doi: 10.1145/3229345.3229401.
- J. Rushby. *Formal methods and the certification of critical systems*, volume 37. SRI International, Computer Science Laboratory, 1993.

- C. F. Snook and R. Harrison. Practitioners' views on the use of formal methods: an industrial survey by structured interview. *Inf. Softw. Technol.*, 43(4):275–283, 2001. doi: 10.1016/S0950-5849(00)00166-X.
- M. H. ter Beek, A. Borälv, A. Fantechi, A. Ferrari, S. Gnesi, C. Löfving, and F. Mazzanti. Adopting formal methods in an industrial setting: The railways case. In M. H. ter Beek, A. McIver, and J. N. Oliveira, editors, *Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Porto, Portugal, October 7-11, 2019, Proceedings*, volume 11800 of *Lecture Notes in Computer Science*, pages 762–772. Springer, 2019. doi: 10.1007/978-3-030-30942-8_46.
- J. Weber. *Automotive Development Processes: Processes for Successful Customer Oriented Vehicle Development*. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-01253-2. doi: 10.1007/978-3-642-01253-2.
- J. M. Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–24, 1990. doi: 10.1109/2.58215.
- C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell. *Experimentation in Software Engineering*. Springer, 2012. ISBN 978-3-642-29043-5. doi: 10.1007/978-3-642-29044-2.
- A. Zaidman, N. Matthijssen, M. D. Storey, and A. van Deursen. Understanding ajax applications by connecting client and server-side execution traces. *Empir. Softw. Eng.*, 18(2):181–218, 2013. doi: 10.1007/s10664-012-9200-5.