The Journal of Systems & Software 199 (2023) 111624

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Testing of highly configurable cyber–physical systems — Results from a two-phase multiple case study☆

Stefan Fischer [a,*], Claus Klammer [a], Antonio Manuel Gutiérrez Fernández [b], Rick Rabiser [b], Rudolf Ramler [a]

[a] Software Competence Hagenberg GmbH, Hagenberg, Austria
[b] CDL VaSiCS, LIT CPS Lab, Johannes Kepler University Linz, Linz, Austria

ARTICLE INFO

Article history:
Received 3 June 2022
Received in revised form 21 November 2022
Accepted 20 January 2023
Available online 28 January 2023

Keywords:
Software testing
Variability
Configurable software
Software product lines
Cyber–physical systems

ABSTRACT

Cyber–physical systems are commonly highly configurable. Testing such systems is particularly challenging because they comprise numerous heterogeneous components that can be configured and combined in different ways. Despite a plethora of work investigating software testing in general and software product line testing in particular, variability in tests and how it is applied in industry with cyber–physical systems is not well understood. In this paper, we report on a multiple case study with four companies maintaining highly configurable cyber–physical systems focusing on their testing practices, with a particular focus on variability. Based on the results of the multiple case study, we conducted an interactive survey with experienced engineers from eight companies, including the initial four. We reflect on the lessons learned from the multiple case study. We conclude that experience-based selection of configurations for testing is currently predominant. We learned that variability modeling techniques and tools are not utilized and the dependencies between configuration options are only partially modeled at best using custom artifacts such as spreadsheets or configuration files. Another finding is that companies have the need and desire to cover more configuration combinations by automated tests. Our findings raise many questions interesting to the scientific community and motivating future research.

1. Introduction

Cyber–physical systems (CPSs) are engineered systems built from and depending upon seamlessly integrated computation and physical components.[1] Such systems are highly configurable. They exist in many variants to support the requirements of different customers and markets as well as to address frequently changing hardware requirements. CPSs comprise many heterogeneous components (mechanical, electrical, software), which can be configured and combined in a myriad of ways.

Research on variability management has been conducted in the field of software product lines for over 30 years (Raatikainen et al., 2019; Rabiser et al., 2018). Case studies with industry have been conducted to better understand highly configurable systems in practice, e.g., regarding modeling and managing variability (Berger et al., 2013a, 2015; Villela et al., 2014; Fischer et al., 2018), used tools (Bashroush et al., 2017), and open challenges (Mukelabai et al., 2018; Berger et al., 2020). Also, many case studies on software product line adoption exist (Martinez et al., 2017).

Many works discuss software testing practices in industry in general (Engström and Runeson, 2010; Kassab, 2018; Kassab et al., 2017; Vierhauser et al., 2014) and software product line testing in particular (Lopez-Herrejon et al., 2015; Engström and Runeson, 2011; do Carmo Machado et al., 2014a; Lamancha et al., 2010). Recently, Berger et al. (2020) reported about a study of twelve medium- to large-scale companies – from domains such as automotive, aerospace, and railway systems – regarding their adoption of variability management techniques. Results show evidence that software product line engineering techniques in general are more and more adopted, but little evidence exists regarding the usage of such approaches for testing. This nurtures the assumption that companies deal with variability in their tests implicitly and not in a systematic manner.

Our aim is to investigate in detail how industry manages variability in their testing activities. We do this in two phases, both with the goal to obtain insights in the state of the art of

☆ Editor: Matthias Galster.

* Correspondence to: Softwarepark 32a, 4232 Hagenberg, Austria.
E-mail addresses: stefan.fischer@scch.at (S. Fischer), claus.klammer@scch.at (C. Klammer), antonio.gutierrez@jku.at (A.M.G. Fernández), rick.rabiser@jku.at (R. Rabiser), rudolf.ramler@scch.at (R. Ramler).

[1] https://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf21551&org=NSF

https://doi.org/10.1016/j.jss.2023.111624
0164-1212/

testing variability in practice and to identify issues and blockers that should be addressed by future research.

In *phase 1*, we conducted a multiple case study with four companies developing CPSs. We first conducted a survey followed by in-depth structured interviews with experienced engineers from these four companies. We analyze the survey results and the recorded interviews in detail and derive initial lessons learned regarding testing of highly configurable CPSs.

In *phase 2*, we aim to clarify the results of the multiple case study and to dig deeper. We conducted an interactive survey with eight experienced engineers from eight companies, four from phase 1 and four additional. We prepared the survey based the results of phase 1. Interactive survey means we not just distributed this survey to our subjects, but we met with them and asked them to complete the survey live (without us watching though). We told them to ask for further information if they are unsure about a question or an answer or if they want to discuss something triggered by a certain question. We arranged these meetings in groups of 2 to 4 people from 2 to 4 companies at a time, allowing participants to also discuss among each other. We took notes on the discussions and questions that were asked. We present interesting findings from both – the completed survey and these discussions – to refine the lessons we learned in the multiple case study in phase 1.

For instance, we learned that variability models are not used in industry, but custom-developed configurators or spreadsheet-based approaches are created and used to manage variability. Transfer from research to industry should be increased when it comes to variability and testing. We learned that more automation is necessary to populate and maintain variability models and to reduce testing effort. We also found that multidisciplinary (variability modeling) approaches are needed in CPSs to deal with knowledge and artifacts from different engineering disciplines.

This paper is an extension of our previous paper (Fischer et al., 2021a) in which we presented the multiple case study (phase 1), however, only including three companies. For this paper, we added a fourth company to the multiple case study and refined the results accordingly. Furthermore, we also conducted an additional interactive survey (phase 2). From both phases we derived lessons learned for researchers and practitioners. In this paper we also discuss background and related work in more detail.

The remainder of this paper is structured as follows. In Section 2, we describe the relevant background and related work of our research. The research approach and the design of both phases are described in Section 3 along with procedures for data collection and analysis. Section 4 describes phase 1, i.e., our multiple case study. We discuss the results and observations in relation to our research questions. We highlight practical challenges and open issues in variability testing as well as threats to validity. Section 5 describes phase 2, the interactive survey. We describe the four additional companies, present and discuss the results of the interactive survey, and clarify threats to validity. Section 6 describes lessons we learned from both phases for practitioners and researchers in the area of testing highly configurable (cyber–physical) systems. We conclude the paper and outline future work in Section 7.

2. Background and related work

We discuss background and related work on configurable software, testing configurable software, and testing CPSs.

2.1. Configurable software

A configurable software system can be intuitively defined as a generic software with a set of mechanisms to implement planned variants so it can be flexibly adapted to different needs. These variants can include different functionality (e.g., enable/disable features), hardware support (e.g., running operating system), or performance (e.g., increase quality of service), among others. There are different mechanisms to provide this flexibility, such as preprocessor directives, control execution in the code (e.g., simple IFs), frameworks with configurations, command-line options, etc.

These possible configurations need to be specified in a variability model (Czarnecki et al., 2012). This model can be informal such as a spreadsheet or a dedicated language or modeling approach. It serves among other to guide the user during system configuration. Variability modeling approaches commonly declare the system variants as features (Kang et al., 1990; Berger et al., 2013b) or decisions (Schmid et al., 2011). Popular commercial feature-model based configurator tools are pure::variants (Beuche, 2016) and Gears (Krueger and Clements, 2018). Academia also has produced (open source) tools (Bashroush et al., 2017) such as FeatureIDE (Meinicke et al., 2017).

2.2. Testing configurable software

A high number of variants hinders the design and execution of test suites, which effectively cover and validate configurable systems (Metzger and Pohl, 2014). Variation points have to be considered to decide what has to be tested and to test if the system is properly built. The reuse of common components or platforms should be also included as part of testing, e.g., reuse of test cases. And, lastly, since the number of variations can grow exponentially, a strategy to avoid redundant testing or to prioritize critical tests is needed. In 2011, Engstrom and Runeson (Engström and Runeson, 2011) systematically analyzed the existing approaches to testing of software product lines, stating that software product line testing was a rather immature area by that date. A number of approaches from the software variability domain address testing by using the variability model to generate the test cases. For test generation, a multi-objective function can support the selection of priority test cases (Hierons et al., 2020; Parejo et al., 2016) based on cost or computation-time. Srikanth et al. (2009) analyze the project history to determine the test cases with higher impact on system failures. In case no such variability model exists, it can be synthetically created based on domain knowledge and mutation techniques for testing purposes (Czarnecki et al., 2012; Lopez-Herrejon et al., 2014; Galindo et al., 2016, 2014). Together with the variability model, the reuse of existing testing artifacts is considered to generate new test cases (Fischer et al., 2019). However, the adoption of systematic testing approaches for highly variable systems requires further efforts depending on the system domain.

2.3. Testing CPSs

CPSs typically require high variability, but in practice these systems are developed using a number of custom-developed and third-party engineering tools, which are often not well-integrated.

Harrison et al. (2016) point out, that there are only few suggestions on how automation systems could be effectively configured and supported throughout their lifecycle.

Several research approaches propose the design of test cases based on the development workflow (Vogt, 2017) or based on the data modeling of the CPS (Humeniuk et al., 2021). Vogel-Heuser et al. (2015) discuss existing challenges in the development of automated production systems (APS) at different development lifecycle stages and point out how to address them. However, there is little evidence of the application of such techniques in industry (Berger et al., 2020).

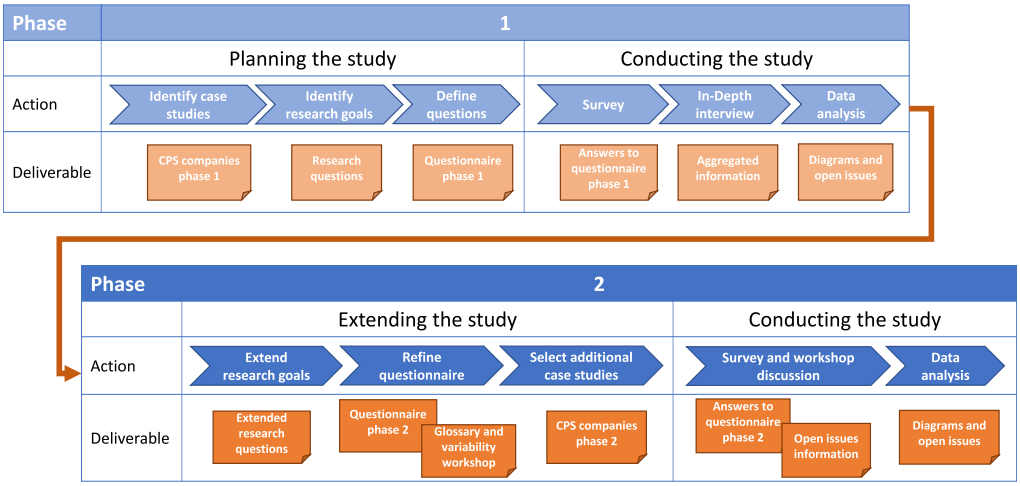


Fig. 1. Case study workflow.

Sinha et al. (2019) review of formal methods for creating reliable industrial automation systems. They conclude that there are approaches to formalize the specification, automate the generation of goals, and evaluate their correctness with respect to the specification. However, there is no evidence of systematic variability management and testing. A number of approaches have been proposed to automate the generation of machine variants and to support modularization. Vogel-Heuser et al. (2021) propose to manage variability in CPSs using the object-oriented extension of IEC 61131-3 (IEC, 2013), but they do not consider other aspects of configuration. Biffl et al. (2017) summarize existing work on multidisciplinary engineering for cyber–physical production systems in their 2017 book. Biffl et al. (2015) also present a model-driven engineering approach for developing, versioning, and binding support for cyber–physical production systems modeled with AutomationML. Eckert et al. (2012) propose the use of specific patterns for application distribution especially to address non-functional requirements. Fay et al. (2015) use model-driven engineering for distributed manufacturing automation systems with design patterns. All of these approaches improve the project development process, but they do not systematically address variability modeling and testing. Finally, existing techniques from software product line engineering are adopted to model the variability but the application of such techniques to testing CPSs has been neglected so far.

3. Research design

Our study of the state of the practice in testing highly configurable systems follows the guidelines for case study research by Runeson and Höst (2009). The concrete steps of this work are depicted in Fig. 1. First, in phase 1, we selected four representative case companies (from our network of industry partners) developing and maintaining CPSs and described our *research questions* 3.1 to study the state of practice. Based on these research questions, we created a *questionnaire* 3.2 that was delivered to the participants from the selected case companies. Details about the companies used as case studies are provided in Section 4. After the survey we conducted in-depth *interviews* 3.4 to solve ambiguities and to discuss open topics. The results of the interview sessions were collected as coded *answers* together with the *aggregated information* 3.6. In a final step we validated and summarized the results, and we identified open issues and avenues for future research. We describe these results in Section 4.

In phase 2, we further analyzed and discussed our experiences from the phase 1 to extend the results. First, we extended our *research questions* 3.1 to learn about the industry practices of managing configurable systems and, specifically, the testing of such systems. For this purpose, we created a new *questionnaire* to focus on the challenges identified in phase 1 3.2. In order to gain a deeper understanding of these challenges, we arranged *interactive sessions* with eight companies (the four companies from phase 1 plus four additional ones). Each interview session has been conducted live via Zoom or MS Teams and included a combination of people from different companies. The participants were encouraged to ask and discuss about the questionnaire live and after answering the questionnaire (without our intervention). Furthermore, questions about current testing strategies and open challenges were also discussed together and among the participants 3.5. We coded and analyzed the results of phase 2, which are presented in Section 5.

3.1. Research questions

Based on the state of the art in testing highly configurable systems (Lopez-Herrejon et al., 2015; Engström and Runeson, 2011; do Carmo Machado et al., 2014a; Lamancha et al., 2010), we considered the following research questions (RQs) for phase 1 of our study.

RQ1-1: Is variability a consideration for all test levels? *Rationale:* The scope of the variability determines in which testing level it must be considered. For example, if individual artifacts are not configurable, the corresponding unit tests are not affected by variability. However, if the units can be composed into different system configurations, variability still matters in integration or system testing.

RQ1-2: How are configurations selected for testing? *Rationale:* A common issue with configurable software is that the combination of interacting features can result in undesired side effects. And, also, the number of possible combinations can be too large to be fully tested in practice. To solve these issues, e.g., different sampling techniques to select configurations for testing have been proposed (Lopez-Herrejon et al., 2015).

RQ1-3: What is the degree of automation in testing? *Rationale:* Test automation is key to have continuous (and short) development cycles, which increase system quality and reduce the effort for manual testing tasks. However, the complexity of configurable systems can be a challenge for the design and implementation of comprehensive automated test suites.

RQ1-3a: Which testing tasks are automated? *Rationale:* The complexity of the development stages affects the costs and benefits of manual testing. While manual testing may be needed for

certain exploratory cases, identifying strategies to automate more testing tasks is beneficial to be able to take full advantage of automated testing.

RQ1-3b: What test environments are required? *Rationale:* Establishing and configuring the setup required for test execution has a significant influence on the testing effort and the design of automated tests. For example, tests may depend on specific hardware components, which may need manual configuration before tests can be run. Digital twins or simulations may help to avoid dependencies on physical devices. However, some tests may still require execution on the actual target hardware to produce representative results.

For phase 2, we considered the following research questions:

RQ2-1: How is variability managed in cyber–physical systems in practice? *Rationale:* During phase 1 we learned that in order to better understand how companies test their highly configurable systems, we must first understand in more detail how they manage variability in general, what variability mechanisms they use, how they represent system variants, and which tools they apply.

RQ2-2: How are configurable cyber–physical systems tested? *Rationale:* Based on a better understanding of variability management practices, we wanted to analyze in detail how companies test their configurable systems and how they consider variability when testing.

When presenting the results of phase 2, we relate these results to the new research questions RQ2-1 and RQ2-2 as well as to the original research questions from phase 1.

3.2. Questionnaire

In alignment with the research questions RQ1-1 to RQ1-3, we created a first questionnaire that was delivered to the four case companies in phase 1. Our questionnaire was based on the questionnaire used in the study of industrial needs and practices for analyzing highly configurable systems by Mukelabai et al. (2018). Furthermore, to adapt and extend our questionnaire we also considered related literature on variability and testing (Lopez-Herrejon et al., 2015; Engström and Runeson, 2011; do Carmo Machado et al., 2014a; Lamancha et al., 2010), the maturity model for SPLs (Ahmed and Capretz, 2015), and the Test Process Improvement model TPI (Sogeti, 2009).

The questionnaire is available online (Fischer et al., 2022). It is divided into four sections. The first section *Generic System Characteristics* includes three questions to delimit the domain and scale of the developed systems. The second section *Variability Management* supports identifying the scope of the variability in the studied case (thirteen questions). The third section *Testing* includes questions related to the proposed research questions (eight questions). Finally, Section 4 on *Professional Background* targets the respondent's role (two questions) and is taken into account for in-depth follow-up interviews.

The questionnaire was reviewed and revised for RQ2-1 and RQ2-2. As we detected some ambiguities in the answers that were later clarified during the interviews, we included a variability glossary as part of the questionnaire in phase 2. The results obtained in phase 1 also indicated that the variability management was not systematically analyzed. Therefore, we adapted the questions in the second section to go deeper into the challenges of applying techniques to formalize variability management (eight questions). We also updated the questions in the third section *Testing* to cover techniques for test reuse and challenges (four questions). The questionnaire used in phase 2 is also available online (Fischer et al., 2022).

3.3. Survey

During phase 1, we provided the questionnaire to the companies as an interactive PDF to avoid privacy concerns regarding online forms. We sent the questionnaire to our contacts at the case companies for internal distribution to potential participants from different departments. Participants answered the questionnaire on their own without any time limit or oversight. Completed questionnaires were returned via email several weeks after distribution. We received a total of 22 responses.

3.4. Interviews

As a follow-up to the survey in phase 1, we performed semi-structured interviews to gain further insights and a better understanding of the responses collected with the questionnaires in the previous step. We used an interview guide (available at Fischer et al. (2022)) with mostly the same structure as the questionnaire. The interviews were held in groups of two or three participants, who were selected by the case companies as representatives for the survey respondents. They included at least one person from each department that took part in the survey. In total, we interviewed 17 participants. We used the results from the survey during the interviews to identify areas in which participants disagreed and to further clarify the reasons for these differences. A subset of the authors conducted each interview either in person or via video conference, typically lasting around one to two hours. We recorded and transcribed all interviews.

3.5. Interactive sessions

For phase 2, we arranged three interactive sessions with a total of eight participants. These sessions included two different parts. First, each interactive session started with a short seminar about variability terms and techniques because we had noticed a lack of common understanding of variability techniques in phase 1. After this seminar, we provided the questionnaire for phase 2 as PDF to the participants to review and answer the questions in a 20 min time window. We allowed participants to ask clarification questions about the questionnaire. After the participants had completed the questionnaire, we immediately started discussions about the questions and answers to identify the main problems in the adoption of variability management techniques and the approaches currently used for testing different configurations.

3.6. Data analysis and interpretation

In both phases, we manually analyzed the questionnaire responses quantitatively by creating statistics and diagrams, and qualitatively by examining the responses to open-ended questions. To analyze the interviews, we used open coding (Corbin and Strauss, 2008) supported by the tool MaxQDA.[2] We created codes based on the structure (sections) of the questionnaires used as basis for interviews and discussions. Subsequently, we identified concepts for each section and introduced them as sub-codes. Two authors coded the interviews independently. The codes were discussed and refined by all authors in several rounds before finalizing them. Our final code system is depicted in Fig. 2. Using these codes, we identified interview responses related to certain topics and used them to answer our research questions. Additionally, we analyzed the interviews for open issues for further research opportunities.

[2] https://www.maxqda.com/

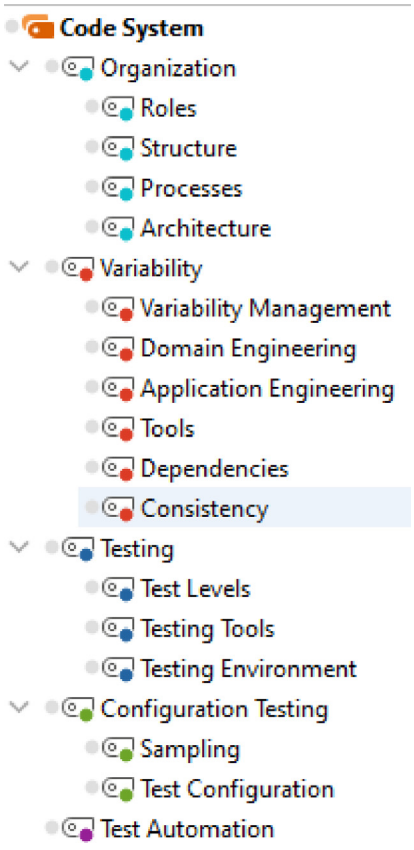


Fig. 2. Coding system used for analyzing interviews.

Table 1
Interview participants.

Part.	Case	Exp.	Role
P01	1	> 10	Developer, Team leader, Project manager, Domain expert, Software architect
P02	1	> 10	Developer, Modeler, Team leader, Project manager, Product owner, System architect, Software architect
P03	1	> 10	Developer
P04	1	> 10	Developer, Team leader, Project manager, Product owner
P05	2	5–10	Developer
P06	2	5–10	Developer, Team leader, Software architect
P07	2	3–5	Team leader, Project manager
P08	2	1–2	Developer (responsible for testing)
P09	2	> 10	Developer, Project manager, System architect, Software architect
P10	2	3–5	Developer, Software architect
P11	3	> 10	Developer, Modeller, Team leader, Researcher, System architect, Software architect
P12	3	> 10	Developer, Project manager, Domain expert, Team leader, Researcher, Software architect
P13	3	> 10	Developer, Team leader, Modeller, System architect, Product manager, Project manager
P14	4	5–10	Developer, Modeller, System architect
P15	4	> 10	Developer
P16	4	> 10	Developer, Modeller
P17	4	> 10	Developer, Team leader, Project manager, Domain expert, Product owner, System owner, System architect

4. Phase 1: Multiple case study

4.1. Studied cases

In the following, we list the case companies from phase 1. As context for our study, we first provide details about the systems that they develop and how they manage variability. Then we describe their testing techniques with respect to test levels (RQ1-1), configuration selection (RQ1-2), automated testing tasks (RQ1-3a), and testing environments (RQ1-3b). We include quotes from some of the interview participants in the case descriptions marked with Pxx identifying the quoted participant. All interview participants are listed in Table 1.

4.1.1. Case 1

The first case company is a manufacturer of a wide range of industry machines. Four representatives of this company participated in our analysis. Three of them are development leaders from different areas and the fourth is responsible for the machine commissioning at customer facilities.

System/product (focus on variability). The machines offered by the company are variations of 4 to 5 base machines that can be distinguished due to the technology used for the drive (e.g., hydraulic, electric, etc.). Due to the number of configuration alternatives, a wide range of versions are available for each base machine. The possible different functionalities define the configuration options. The software is developed for Programmable Logic Controllers (PLCs) and for Human Machine Interfaces (HMIs) with a touch panel. The software for both parts is designed with loosely coupled components that can be reused and adapted independently to fulfill specific project requirements. Over 700 pre-developed components can be reused to create numerous system variants.

The configuration of a system according customer requirements starts in the ERP (enterprise resource planning) system. The options made available in the ERP system are linked to respective components and HMI modules. In the ideal case, the software can be configured automatically from the set of selected options. Additionally, the company provides customer-specific extensions for their system, which lead to additional development and testing effort (RQ1-3b). Moreover, constraints between options are not modeled in the ERP system, but rather are expert knowledge of the personnel involved in building (and testing (RQ1-2)) customer-specific configurations.

The core assets of the system are the mandatory and optional modules, which can be configured automatically and which are maintained by the development department. In SPLE terms, the development department performs the domain engineering of the system. Customer-specific requirements are handled in the engineering department, usually by adapting the pre-developed modules. They handle commissions as well as what in the SPL literature is referred to as application engineering. In the best-case scenario, the engineering department can use the standard modules directly and generate the software for a particular machine. However, in case some of the customer requirements are not covered by the pre-developed modules, the engineers extend them into specific variants. Whether an extension eventually is included in the set of standard modules is decided on a case-by-case basis, e.g., when several other customers also request the extension.

A machine can optionally be equipped with a robot, which is controlled by the machine's PLC. The robot is developed as separate sub-system. It can also be sold stand-alone without being attached to a machine, which then requires its own PLC.

The company has developed its own proprietary configuration tool for robots, which contains hard-coded constraints to avoid invalid configurations (affecting also RQ1−2). In contrast to the other parts of the machine, robots use pre-processor annotations in the source code that are processed by the configurator tool.

Finally, the machine and the robot both further use run-time configuration options, which add another layer of complexity. Some of these configuration options are defined at commissioning according to hardware-specific parameters (e.g., movement speed). Other options can be altered by the customer, as they control the behavior of the system (e.g., machine operation sequences).

Testing of variability. The high variability of the system makes it infeasible to test all possible variants that can be configured. Therefore, *the company tests specific configurations, which are selected based on domain expert knowledge* (RQ1−2). Configurations are selected with the aim to cover as many options as possible as well as interactions between options that may cause interaction problems. However, due to the strong emphasis on modularization and loose coupling of components, the interviewees reported that interactions are not a critical issue. P01: ''We have few problems with combinatorics, more with the options themselves.'' During development, thus, the *tests at the unit test level are largely performed independently from specific configurations* (RQ1−1).

On the *integration* test level (RQ1−1) and above *tests are related to specific system and run-time configurations, on which they can be executed*. Annotations are used to define test sets that run on specific configurations (RQ1−2). *On all levels, there exist some tests that can be executed independently of the configuration.*

Each software version has to pass an acceptance test process by a separate quality assurance department, which performs system tests on real (physical) machines in three different manifestations (RQ1−3b). They choose the tested configurations based on changes linked to the specific version (RQ1−2). These tests often include installing new required hardware on the test machines. P02: ''Obviously, it would be too expensive to have a variant of every machine in every hardware configuration available.'' This acceptance test phase lasts for about two months. A large set of manual system tests (RQ1−3) are performed. All of them have to pass before the software version can be officially released.

Automation and test environment. Test automation (RQ1−3) is realized in form of automated tests that are executed during a nightly build, which is based on Jenkins[3] jobs. These tests are implemented using proprietary testing frameworks to test the PLC code and JUnit as well as TestNG to test Java components. For testing the robot the test configuration can automatically be generated before the corresponding software tests are executed (RQ1−3a). *Approximately 10% of the tests are automated (i.e., executed automatically in continuous testing). The rest are manual tests* (RQ1−3). System level tests are mostly manual tests.

To assist development and testing, *different test environments* (RQ1−3b) are utilized. The most widely used environment is a *virtual machine that simulates the entire system* including all hardware components. This simulator is used heavily in the development and design phase as a virtual twin. The majority of the tests can be executed in this virtual environment. Additionally, developers and testers have access to *Hardware In the Loop (HIL) environments* including parts of the real machine such as the HMI panel and the control unit for running PLC code. Other machine hardware is simulated, e.g., any mechanically moving parts. Robot tests utilize a similar HIL environment. However, the entire hardware of the robot is available and tests are performed in combination with a simulated machine. Finally, as mentioned above, the quality assurance department tests on real machines in the same configuration that is delivered to the customers.

> ***Summary:*** *Variability is a consideration on the integration test level and above and not for the unit tests (RQ1−1). The selection of configurations to test is done by experts (RQ1−2). This is challenging due to missing modelling of constraints and different views of the variability in different sub-systems. Only about 10% of tests are executed automatically in a nightly build (RQ1−3). Moreover, the configuration to test can be automatically set up (RQ1−3a). The tests are executed in a variety of environments from a simulation in a virtual machine, a HIL system with some real hardware and simulations, and finally on the real machine (RQ1−3b).*

4.1.2. Case 2

The second case relates to a company developing heavy-duty lifting systems. We interviewed six representatives of this company. Five are from different product lines' software development teams. The sixth interviewee is a developer of the shared run-time system, which is the common basis for these product lines.

System/product (focus on variability). The company maintains different product lines for different application domains and markets. Each product line is highly configurable. All of them are built on top of the same run-time system and all have a similar system architecture that is made up of one central control unit and a few decentralized control units, which are connected to actuators and sensors. The decentralized control units register themselves at the central control unit. The communication between control units is based on a CAN bus, with the central control unit performing all calculations and controlling the connected components. The number of sensors and actuators as well as the behavior of the system can be customized or extended by additional features. The core software system is developed to support all possible extensions upfront, so the software does not have to be updated due to a specific configuration later. P09: ''We have one software for the whole product range that is parametrized according to the size and the purchased features.'' One reason for this approach comes from the need for certification to fulfill safety regulations. The system is certified according to criteria of the ISO/IEC standard 61508 (IEC, 2010), which requires additional time and effort in development and testing. Changes and updates to the system after it has been certified have to be minimized to avoid the need for costly re-certifications.

The variability of the system manifests itself in more than 2.000 configuration parameters. By the interviewees' estimation, roughly 200 of these parameters are related to the main hardware components and are linked to product options in the ERP system. Possible product configurations are maintained in a database. However, not all dependency restrictions between configuration parameters are specified in the ERP system, because several hundreds of variant combinations are possible (RQ1−2). In addition, the complexity and variety of the possible configurations also differ between the different product lines and depend on the degree of project-specific development.

Specific features of the system are set by configuring parameters to a predefined set of values. However, in some cases – for instance, when new features are introduced and dependencies are still unclear – configuration problems are detected late, i.e., at machine assembly time. The company tries to eliminate such configuration problems with *inconsistency checks that are*

[3] https://www.jenkins.io/

implemented in the firmware of the control unit as hard-coded requirements (RQ1–3a), but no formal description of configuration dependencies exists for automated reasoning.

An abstraction layer in the architecture enables the use of different hardware platforms (control units) and different run-time operating systems. Therefore, there is only one control software for every supported control unit, which uses the abstraction layer to provide support for all product variants (RQ1–1). Hardware specific settings (e.g., the number of available I/Os, the maximum number of supported CAN-Nodes, or available processing resources) are configured statically in header files (RQ1–2).

The different product line teams develop their own applications on top of the core system. These applications provide features implemented in one or more software modules, which are currently not shared between different product lines. Variability comes from being able to configure different modules with one another (RQ1–1). Single features and software modules can be activated dynamically via parameter settings (RQ1–2). The proprietary configurator and diagnosis tool supports the correct setup of the parameters. This tool is available also to customers building on and integrating the provided products in their solutions. Moreover, the configurator and diagnosis tool is also a central tool used during testing and for problem analysis (RQ1–3a).

As a summary, the system provides two main configuration options: static control unit configuration at compile time (i.e., pre-processor annotations) and the dynamic configuration (i.e., predefined parameter settings) of applications to support all options of one product line (RQ1–1). The main purpose of this approach is to keep the number of different software versions undergoing certification low, while supporting as many variants as possible (RQ1–2).

Testing of variability. The company follows the V-model development process. It ensures consistency of parameter values with necessary requirements documents and testing at every level, from module and integration tests up to requirements validation. Additionally, the company uses naming conventions for configuration parameters and other rules that should prevent erroneous configurations upfront. Different *configurations are not tested explicitly at the unit testing level* (RQ1–1), since at this test level they are just simple parameter values. Configuration combinations are tested only to achieve full code coverage. For static configurations (i.e., pre-processor annotations), one maximum configuration setting is used in unit testing. P05: ''We test with the maximum configuration and the rest turns up at integration and component testing.''

Selecting appropriate configurations for testing at the integration and system level is partly defined in the requirements document. The executed tests are recorded and documented. They use the tool JIRA *to trace the requirements to tests*. Testers try to identify configurations that cover many of the aspects described in the requirements documentation to verify the system's correct behavior (RQ1–2). In addition to the in-house system tests for each release, *each individual overall (integrated) system undergoes additional final tests and certifications to guarantee the safety of the product in its final working environment* (RQ1–3).

Automation and test environment. Testing and test automation are driven by safety requirements and regulations as well as the corresponding certification process (RQ1–3). For unit testing of the software modules the tool Tessy[4] is used, because it is an accepted tool by the certification authority to prove the required 100% level of code coverage (RQ1–3a). The tests are executed by a Jenkins build server and the quality of the tests is supported by a rigorous coverage analysis and a set of additional mutation tests (RQ1–3a).

At the higher testing levels, the *degree of automation differs between the product lines* (RQ1–3). P07: ''Unit tests work wonderful with Tessy. On the other hand, integration tests with some modules are more difficult, because there is support missing.'' Manual integration tests exist, which are realized with a custom-built test setup. This test setup uses the real control unit and provides manual switches and rotary controls to simulate the state of the machine by setting inputs and outputs accordingly (RQ1–3b). This further enables testers to perform tests that would be dangerous on the real equipment and could lead to damage and injuries (e.g., trying to lift outside the safety weight boundaries). Other product lines already utilize an automated testing approach at the integration level (RQ1–3a). They define test scripts that can be executed by the tool CANoe[5] (RQ1–3b).

In recent years the company started to implement HIL systems (RQ1–3b) to ease automation of integration and system level testing (RQ1–3). Additionally, the company invested much effort to run functional system tests within a Windows-based development environment (RQ1–3b). Thus, not every developer requires a test setup including a PLC for testing anymore. Instead the software is compiled with a Windows C-compiler and runs on a personal computer. It helps debugging and locating errors at an earlier stage.

Although certification requires the tests to be run on the actual hardware, *virtualization and simulation is seen as an important step to be able to automate testing and to test the even more complex and flexible products of the future thoroughly*. The most important challenge within the context of real-time operating systems is the different time behavior in the virtualized setup (RQ1–3).

> ***Summary:*** *The variability of the system comes from the combination of different modules and therefore is a consideration only for integration test level and above, and not for unit tests (RQ1–1). The tested configurations are selected by experts with the support of their requirements documentations (RQ1–2). The degree of test automation varies between different product lines and test levels but is constantly improved with introduction of better HIL systems (RQ1–3). Unit tests are automatically executed with a certified testing tool, which they are also starting to use for integration testing on the HIL systems (RQ1–3a). Additionally, they use a proprietary diagnosis tool to automatically detect inconsistencies and configuration issues. Tests can be executed in simulation environments on the developers' PC, on real control units with manual interactions to "simulate" hardware, on their HIL environments, and some tests that have to be run on the real hardware (RQ1–3b).*

4.1.3. Case 3

Our third industry partner is a provider of building security solutions. The company offers planning, development, and application of access control and monitoring systems. We interviewed three members of the research and development department, who all have software development background and over 10 years of experience with the system.

System/product (focus on variability). The case company supports systems of different size ranges from small installations to large distributed systems, which integrate diverse external systems and hardware (e.g., control of door-opener systems, card readers, fire detectors, etc.) (RQ1–2). Therefore, the system configuration

[4] https://www.razorcat.com/en/product-tessy.html

[5] https://www.vector.com/int/en/products/products-a-z/software/canoe/

primarily dependents on the used hardware components and the other systems that need to be linked. Besides that, the system also contains optional features that have to be licensed separately. In addition to the installation size, the integrated hardware, and the optional features, the product including its software can be configured to conform with the customer's workflow (RQ1–2). The two main parts of the product are an access control system and a video monitoring system. These two parts can be purchased individually or together. Technically, however, these two parts are heavily integrated. The variability on the code level is realized as conditional execution (i.e., if-conditions). The modularization of the software system into components has been done from a purely technical perspective and independently of features or options, which can span over several modules.

The system is configured using a *proprietary GUI-based configurator tool*. Hardware configurations, optional functionality, and runtime behavior configurations are all done via this configurator tool. Constraints among configuration options are hard-coded in the source code of the configurator tool. Various modules provide a meta-data description, which is used to build a generic configuration GUI. However, the constraint specification focuses more on the runtime behavior configuration since this configuration can be changed by the end-customer. Other parts of the configuration are not as fully defined in the constraint specification and require an experienced technician to adjust the configuration at system setup. Configuration settings are currently stored in a database and partly in files, but it is the goal to provide a one-stop-shop solution for the maintenance and commissioning team to configure the system. P11: ''Our aspiration is to provide a configuration tool that can be used by a trained person to be able to parameterize the whole system in all of its facets.'' For large facilities, it is sometimes still necessary to use the scripting capability of the system to implement customer-specific requirements (RQ1–1).

In the most cases, the software can be configured and deployed to a customer site out of the box due to the available variability mechanisms. Nonetheless, customers occasionally need new requirements not supported by the existing features and configuration options. In such cases an evaluation is performed to decide if the requirement is a valuable addition to the existing system. If this is the case, it will be added in an upcoming release and support is provided in all future versions of the system. P12: ''There is no project-specific software. If the project is finished, possible customer-specific software becomes part of the product.''

Product development is driven by requirements coming from single customer projects. SCRUM with three-week sprints is used as development process to continuously advance the product. Once a month a service release is published, which is preceded by a regression testing phase. Each release contains the entire functional range of the system. Since the hardware is very durable, it is almost impossible to discontinue support in the short term.

Testing of variability. The company employs a variety of test types. Unit tests are created as part of the development process and can be performed independent from the configuration. P11: ''Unit tests should not have any dependency to a system configuration. All other tests depend heavily on the configuration.'' (RQ1–1) Integration tests also depend on other software modules. These tests are developed to be executed automatically within the build infrastructure (RQ1–3a). The setup of the required system configuration is part of the test setup and was reported as a major challenge by the interviewees (RQ1–3a). P11: ''Actually the creation of the appropriate configuration or parameterization makes up the majority of the test effort.''

The *configurations that are tested are selected mainly based on experience*. They include relevant combinations of tested features and configurations that are typical in practice. *Configuration options accessible by the end-user to change the runtime behavior via the user interface are covered more thoroughly by automated system tests* (RQ1–3a). Currently, tests for a given functionality cannot be automatically executed on several different configurations (RQ1–2). The tests often rely on specific hardware and software dependencies such as a particular database and message bus (RQ1–3b), which cannot be automatically setup or re-configured.

The ever-growing set of features makes it infeasible to perform all regression tests for each service release. Instead, *regression tests are only performed on the core functionality and on selected system parts that have changed since the last major release* (RQ1–2). Additionally, *new functionally is tested manually by a dedicated tester as part of the development process.*

Automation and test environment. The company uses *a variety of automated testing frameworks*. On the lowest test level, frameworks for automated unit testing are used (RQ1–3a). These unit test frameworks are also used to realize some of the automated integration tests. Additionally, the company utilizes SpecFlow[6] to specify and execute tests following a behavior driven development approach. Automated GUI tests are realized with Ranorex[7] and Selenium[8] (RQ1–3a). However, the participants from the case company reported *difficulties in bug localization when a failure occurs in one these testing frameworks*. Automated tests are continuously executed on a nightly basis or over the weekend, depending on how time-intensive the execution is.

No special test environment is required for unit testing. Unit tests are commonly executed directly from the development environment (RQ1–3b). In contrast, integration tests do require other software systems to be present (e.g., database or message bus) or even hardware components (e.g., card readers, alarm system, cameras). To mitigate the difficulties associated with these dependencies, the company tries to *simulate as much of the hardware as possible* (RQ1–3b). Nevertheless, a large variety of third-party hardware has to be supported. Not all of that can be simulated and, thus, *some tests still require to be connected to the actual hardware*. The setup of specialized custom-made hardware is particularly challenging.

> ***Summary:*** *Unit tests can be performed independent from any configuration, but integration and system tests are developed for specific configurations (RQ1–1). System configurations are very dependent on specific installations and can be distinctly unique to them. Experts select the configurations to test based on changes in the current version (RQ1–2). The degree of automation is already very high in this case (RQ1–3). Tests on all test level are executed automatically in their continuous integration system (RQ1–3a). Challenges come with the automatic configuration of the system, which makes up a large part of the test effort. Moreover, are their tests not configurable to test a different variant. They simulate different devices to execute their integration and system tests, but because of the huge variety of supported hardware the real devices are still often required to run tests (RQ1–3b).*

4.1.4. Case 4

The fourth case is based on a company producing state-of-the-art boilers for individual heating systems using renewable energies. Their product range includes wood chip heating, pellet heating, air/water heat pumps, and solar systems. We interviewed four company members involving the development

[6] https://specflow.org/
[7] https://www.ranorex.com/
[8] https://www.selenium.dev/

leader, a software developer, a test bench engineer, and a product owner. All of them have a software development background and at least five years of experience within the heating domain.

System/product (focus on variability). The variability of the produced boilers is mainly handled by introducing abstraction layers and by using configurations. A central code base provides the core functionality for all of the three main product lines, i.e., pellet, wood, and wood chips. This code basis is built on top of a real-time operating system (RTOS) that is separated via an abstraction layer. An additional hardware abstraction layer ensures vendor-independence for the product line-specific control applications. These applications rely on the common base. They are compiled and packaged into a single software product — one for each product line. Similar to the common base functionality, the user interface (UI) is shared between all product lines. Whether the UI shows the controls relevant for pellets or wood chips is configured at commissioning (RQ1–2).

One of the most important attributes of a heater is its heating performance. The performance class is defined by setting hardware dual in-line package (DIP) switches. The same method is used to enable optional features (RQ1–2) like suction feed or automatic ash removal. However, the overall goal is to keep the number of different software and hardware configurations in the field as low as possible (RQ1–2).

The core development team consists of about five developers. The development approach is inspired by an iterative waterfall process. There is no fixed cycle for new software releases. For example, the software for the pellet product line has been kept at the same version level for over two years. During this time, new product lines have been developed, which required changes in the common software base. The new functionality has been reintegrated into the base software system for all product lines after it has been proven to be stable. Thereby, any changes affecting safety-critical features or the burning process have to be approved by an independent technical inspection agency. Thus, such changes are kept to a minimum. Also, the software of delivered heating systems is rarely updated. The only exception are cases where updates are required due to regulatory changes (RQ1–2). The company does not force customers to update their installations. It is at the discretion of the local service and support contractor to update a system if necessary. During an update, the complete software is updated including all features of the product line.

Testing of variability. The software system under test contains all features as well as all optional equipment characteristics. Most of the variability of the system is determined by its physical characteristics. The existence and correct integration of optional features has to be tested. At the unit level, variability is considered in form of switches and variable values defined by the active configuration. Feature combinations and the intended behavior are considered at integration and system level (RQ1–1). Most of the effort in testing lies in testing the expected behavior in case of a malfunctioning system component, e.g., a temperature sensor which provides wrong values. The company follows a risk-based approach in selecting relevant tests and configurations for execution. P16: ''Because of the complexity it's impossible to check the whole software package at each change. With manual testing as we do it now, it's absolutely hopeless.'' The tested scenarios are defined depending on the variants they consider as most critical (RQ1–2). About one third of the possible configuration parameter combinations are tested.

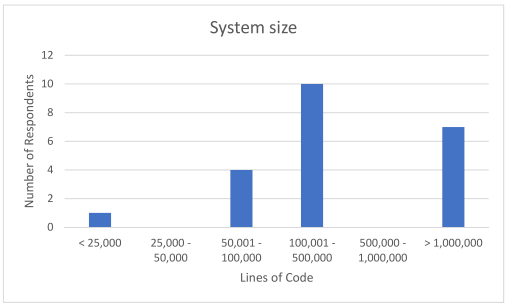
Automation and test environment. In the past, only few automatically executable unit tests or integration tests have been available (RQ1–3). Currently, tests are developed as part of the development of new features. The development of these tests is supported by dedicated test applications, which allow to testing individual components in isolation. In the later phases of development the implemented features are tested together with other functionality as part of so called function groups. Such function groups incorporate real hardware and hardware simulators to allow focused testing including various failure modes. In addition to testing individual function groups there are test setups targeting the correct interaction of function groups, e.g., setups with more than one boiler. All test cases are documented in a test specification (RQ1–1). However, there is no defined process how to create and execute test cases. P16: ''It's a lot of experience how I test and how I describe [test cases].'' In general, testers try to perform testing very close to the real hardware, which makes test automation difficult. Therefore, test environments including simulations exist for every product line. P16: ''We are nearly able to simulate a real object.'' Still, however, the tests in the test environment work without a flame. So after these tests are successful, real operation tests are performed at a test bench. The test bench is fully automated (RQ1–3b) and is used for long running tests (RQ1–3a). But since these tests do not scale and require an operating boiler, the company also utilizes field tests at the customer site. Before new products are released, they are subject to field tests for at least one heating period. Parameter values of test installations in the field are periodically recorded and analyzed to detect weaknesses that cannot be observed at the test bench, e.g., because they may be related to a specific system, fuel, or operator (RQ1–3a).

> ***Summary:*** *Configuration is considered on the unit test level in form of variables to achieve a high code coverage, and on the integration and system test level to test specific system configurations (RQ1–1). To reduce complexity the number of configuration options is kept low. An expert selects the configurations for testing (RQ1–2). Currently only few of their tests are automated, but there are efforts to improve that (RQ1–3). To cover unforeseeable behavior in real installations, the company employs field testing by automatically analyzing data from selected installations (RQ1–3a). Because testing processes that require physical temperature changes can be time consuming, they use simulations that mimic the behavior with speed up time to reduce the required testing time (RQ1–3b). They test long term behavior of the system with an automated test on an environment with a real boiler to consider real physical conditions.*

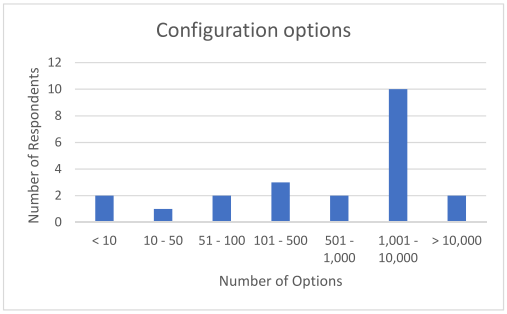
4.2. Results and discussion

In this section we discuss differences and commonalities in how our partner companies handle variability and testing. We refer to the four case studies as CS1, CS2, CS3 and CS4 respectively. Table 1 lists the participants of the interviews from phase 1 showing the company identifiers, the participants' experience, and their roles within the company. We discuss open issues and challenges to further improve testing processes in relation to our research questions. Additionally, we highlight these open issues in form of open questions and describe how we believe they can lead to important future research contributions in *boxes* as part of the presentation of the results. Finally, we also discuss threats to validity.

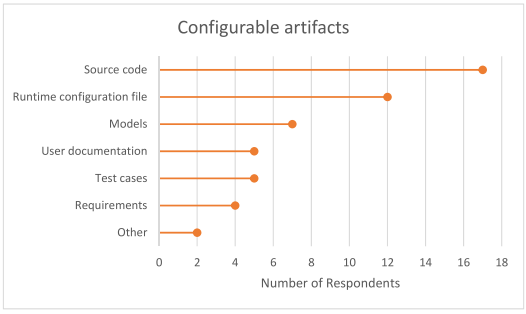
Regarding the organizations involved in the different cases, 13 participants indicated that their teams had less than 10 members and 8 participants answered that the teams had 10 to 50



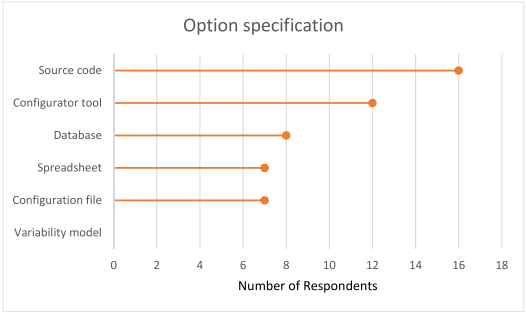
(a) System size.



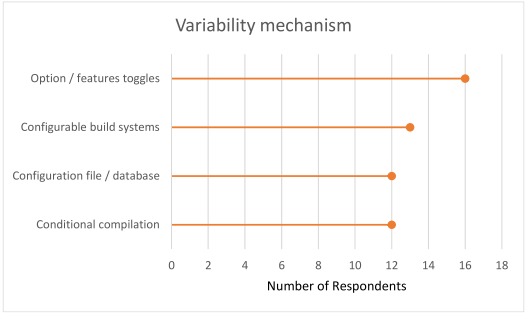
(b) Number of configurations.



(c) Configurable artifacts.



(d) Option specification.



(e) Variability mechanism.

Fig. 3. Results from phase 1.

members. The characteristics of the studied systems are summarized in Fig. 3. Results for questions with only one possible answer are depicted with bar diagrams and those with multiple answers are depicted with lollipop diagrams to visualize more clearly the comparison of options. The numeric intervals for system size and the number of configuration options are based on estimates and experience from the interviewed participants. Along with each response option, we report the absolute number of respondents (out of the 22 respondents) that gave that answer. The number of respondents is indicated in the *x*-axis or *y*-axis depending on the chart type. Although the size of the teams in all cases is smaller than 50 people, 7 participants answered that they are involved in projects with more than 1M lines of code (Fig. 3(a)). Furthermore, 2 participants answered that their systems have more than 10,000 configuration options and 10 respondents answered between 1K and 10K configuration options (Fig. 3(b)). According to the participants, multiple different artifacts need to be configured, not just source code (Fig. 3(c)). Several different variability mechanisms are used (Fig. 3(e)). However, *none of the participants reported that a variability model is applied in their system* (Fig. 3(d)). Rather, *constraints between configuration options are hard-coded* in a proprietary configurator tool or in the ERP system *or constraints are considered domain expert knowledge and handled manually*. Moreover, *test cases are rarely configurable*. All companies reported that they develop tests for specific configurations.

A key difference between the studied companies is how they deal with customer-specific requirements and product line evolution. CS1 provides customer-specific machines for which they reuse configurable components, but commonly they also develop project-specific code that is not re-integrated into the product line. If similar requirements arise in a future project, they fall back to a clone-and-own process (Dubinsky et al., 2013). On the other hand, CS2, CS3 and CS4 do not develop customer-specific software. Participants from these three cases reported that they add

functionality to their configurable core if it is deemed a valuable business decision, for instance, if enough customers request it.

4.2.1. RQ1-1: Variability and test levels

All companies test on all levels, even though CS2 mentioned issues regarding integration testing. On the different test levels, the impact of variability is very much dependent on the system design. *Variability mostly plays a role on the integration, system, as well as acceptance test levels, and not so much on the unit test level.*

How to keep track of test coverage of different configurations at different test stages? Different coverage criteria are relevant at different test levels. Branch coverage might be of relevance at the unit test level, while feature coverage is of interest at the acceptance test level. An overview of the achieved test coverage for every internal release is necessary to check and control testing activities.

Due to the use of pre-processor annotations, only CS2 reported a configuration impact on their unit tests from their hardware abstraction layer (static configuration). Unit tests are executed with a single setup that tries to cover as many configuration options as possible. The main reason is that the applied unit tests are not flexible and, thus, they cannot be reused for different configurations.

Similarly, all companies indicated that they use integration and system tests that can only be run on a single specific configuration. To be able to reuse these tests for different configurations, better support for creating configurable tests or support for automatic reuse would be required (Patel and Shah, 2015; Fischer et al., 2019) .

How can tests be reused more effectively? Applying tests to more configurations can help detect unwanted interactions between configuration variables. However, currently this requires a significant amount of manual work. Research on automatically configuring or reusing tests is required to improve this (Patel and Shah, 2015; Fischer et al., 2019)

4.2.2. RQ1-2: Configuration selection for testing

We found that *the standard practice in all case companies is to select configurations for testing based on individual knowledge and experience from developers and testers*. One reason for this is that *none of the companies has a variability model* (see Fig. 3(d)) that could be used as input to a systematic sampling strategy. Another reason is that sampling methods for huge configuration spaces still suffer from scalability problems. *Even with a systematic sampling strategy, there would be too many configurations to attempt to cover sufficient interactions*. Consequently, it is usually impossible in the studied cases to test the entire sample of configurations (Acher et al., 2012; She et al., 2014).

What is the best way to adopt variability models for an existing system? The dependence on individual developers' knowledge to manually pick the appropriate configurations for testing is becoming a major concern since the complexity of the tested systems is steadily increasing. As a result, methods for reverse engineering variability models are required (Acher et al., 2012; She et al., 2014).

CS2 and CS3 additionally manage the available configuration options to add or remove functionality with additional run-time settings. There is no obvious distinction between software configurations in accordance with the current system variant and licensing with configurable parameters that enable the end user to adjust definite behaviors.

We also observed that the *companies do not analyze test coverage related to different configurations*. Some configurations are only used in testing a specific use case. As a consequence, interaction issues with other functions may not be detected in testing.

Finally, companies sometimes combine static (e.g., pre-processor annotations) and dynamic (feature toggling) variability techniques. In addition, sub-systems developed by different teams or partners may even use different languages (Kim et al., 2011; Nguyen et al., 2014).

How to identify possible interactions? In order to limit the number of configurations to be tested, methods supporting the combination of different languages and variability mechanisms are important for identifying interactions in the source code (Kim et al., 2011; Nguyen et al., 2014).

4.2.3. RQ1-3: Test automation

More testing effort is required when testing a larger number of configurations. This increase can only be achieved by more automation in testing.

The *studied cases show a varying degree of test automation. Even within a single organization, there are product lines with different degrees of automation and different technologies*. As the *companies still perform mostly manual testing*, a large number of configurations require a substantial testing effort. Therefore, some interviewees would like to automate current manual tests. Tools are available to record and replay user actions for the GUI (Garousi et al., 2017), web browser (Sen et al., 2013), or Android applications (Lam et al., 2017). However, the application of such tools to CPSs with hardware dependencies is not sufficiently supported (Thummalapenta et al., 2010).

How to record and replay manual integration tests? Integration tests can be automated to a large extent by means of scripts. However, the large amount of test data for configurations or the creation of multiple hardware simulations make the creation of these scripts error-prone and time-consuming. This problem also tends to increase as systems evolve with new variants and options and tests have to be frequently maintained. The creation of test cases can be aided by using system trace data (Thummalapenta et al., 2010) More research on how to reliably trace configurations and system states for generating integration tests is required.

RQ1-3a: Automated test tasks. Tools for automating unit tests are widely available. Thus, *most test automation is related to unit test execution*. The case companies apply a variety of unit testing tools for this purpose. Unit tests are also reported to be less affected by software variants.

Every company creates and maintains tests for individual configurations by hand. The tested configuration is prepared as part of the test setup. According to CS3, the preparation of a test configuration is a major effort. This finding is confirmed by CS2 where configuration files required for testing have to be created manually. As a result, many of the available features are only tested in a limited number of scenarios.

In addition, all companies confirmed that they face difficulties in reusing tests for different configurations. Only in very few cases the tests can be adapted to other configurations (Tzoref-Brill and Maoz, 2018).

How to co-evolve software and test cases? Tests are highly dependent on the configuration and may need to be modified as the system evolves. Research on solutions to reduce the differences in tested configurations between revisions could help to reduce this effort (Tzoref-Brill and Maoz, 2018).

Some interviewees talked about the *difficulty of reproducing complex system conditions that result in a test failure*. This is especially difficult when a high variety of different components can impact the functionality under test (Li et al., 2021; Wang et al., 2019).

How can complex system states be recreated? For debugging, the root causes of the failures have to be analyzed. Therefore, improving the support to record and recreate the internal state of the system when a failure occurs would be very beneficial. Research in root cause analysis is important for this (Li et al., 2021). However, applying these approaches to CPSs is challenging due to missing support of existing trace engines for hardware-near components, like PLCs (Wang et al., 2019).

RQ1-3b: Test environment. Automation can be more difficult in test-driven environments. Virtualization and simulation are very important, especially in the context of CPSs. In particular, the manual test environments in CS1 and CS2 are hindering automation. The availability and setup of the hardware needed to test different software configurations is an important challenge. To overcome this difficulty, all companies use simulation to varying degrees. However, *it takes a lot of effort to create simulations for all the required components*. This effort can be immense for highly specialized components that customers do not commonly require. In addition, in cases like CS2, *the certification of safety-critical systems requires testing with the real target hardware*. However, some test scenarios (e.g., testing the behavior under hardware overload) need simulations to safely run them (Rafique and Schneider, 2020).

How to best minimize dependencies to hardware during development, while being able to integrate the real hardware if necessary? Real hardware may be required in development and testing. Therefore, it is important that the development ecosystem can easily switch between simulated and real hardware components. Research in using standardized hardware abstraction layers could help improve this (Rafique and Schneider, 2020).

Additional field testing is applied in CS4 since in-house testing can still miss specific faults and long-running test scenarios suffer from scalability issues. Field testing allows the company to monitor systems installed at different customer sites and to subsequently analyze the collected data to evaluate the correct behavior of the system is real usage scenarios (Mossige et al., 2017).

How to use the test environment most efficiently? Companies developing CPSs invest a lot of resources to develop simulations and HIL systems that allow focused testing of the computational parts while simulating the hardware dependencies. HIL systems support execution of integration and system tests but testing at other levels may still require different solutions. To optimize the use of these test environments, research in scheduling the execution of tests on them is useful (Mossige et al., 2017).

The control software of CS1, CS2, and CS4 runs on PLCs (or other dedicated CPUs in case of the HMI of CS1). Testing needs to be adjusted to these specific hardware platforms. For that purpose, the companies use HIL test benches to run the tests of the software components while simulating the mechatronic components. However, in CS2 we observed that some departments still use legacy HIL systems where sending test inputs cannot be automated. In this case the tests have to be executed manually by simulating hardware signals via switches and dials on a dashboard prepared for testing.

How can test environments be created/set up/adapted for a specific system configuration? Preparing system configurations is a very important step in testing and it should be considered when designing the test environment. Automatic configuration of the test environment is needed for many test activities.

4.3. Threats to validity

The main threats to validity for empirical studies are the following:

Construct Validity. As the background and experience of each participant are individual, the responses are subjective. To avoid misinterpretations of terminology, general terms from the case study domain, such as configuration choice or configuration specification, have been used. Participants from different areas and different business domains were also interviewed to gain a broad perspective. In addition, the following interviews allowed us to clarify some ambiguities. In this phase, the questions and interviews were also conducted in German to avoid any semantic leap with a non-native language.

Internal Validity. Another threat is our own bias when evaluating the answers. To overcome this threat, at least two authors transcribed the answers, and the other authors cross-checked the results. Another threat is the selection of participants. For that purpose, we selected the company partners but delegated to them the distribution of the questionnaire to the relevant practitioners with enough experience and from different departments.

External Validity. For the study, we considered companies of different sizes and industry sectors that develop highly configurable systems. However, we only considered four companies in this phase, which can affect the findings. Furthermore, as all of them are Austrian companies, a cultural bias could have been introduced. However, all four companies are international companies, which have offices in different parts of the world and sell their products globally. Developments frequently happen within international teams, distributed between different locations.

5. Phase 2: Interactive survey

In our follow-up to the multi-case study from phase 1, we held three workshops with representatives from eight companies. Four of the companies were also part of the first phase. The other four companies we contacted to be part of phase 2 to involve a wider range of experiences from different domains.

5.1. Company descriptions

All eight companies in our study work with CPSs. In the following we provide a short description of the four companies, which were not already part of phase 1.

Company CS5: This company is working in the complete supply chain of plant building. Each plant is configured to adhere to different customer requirements, existing installations, different hardware providers, or integration with third-party components. They currently have a platform that includes the supported variants in the form of modules. Templates are used to generate project software artifacts from modules. These artifacts have to be extensively adapted according to project-specific requirements. Common modules are tested separately both with simulations and with experimental on-premise validations. Project customizations are tested as part of the project testing.

Company CS6: The company produces fire-service vehicles and equipment. System parts are reused over an assortment of products and have to be configurable to meet different requirements. A vast set of configuration options is specified in an

Fig. 4. Option specification and variability mechanism (phase 2).

in-house database that is used by their custom-developed configurator tool. The configurations to test are selected by experts to execute tests on option combinations that are known to influence certain test scenarios. Currently, their system requires manual configuration for testing. For improving their test automation, they are working on a digital twin, that can be automatically configured and test an arbitrary number of configurations.

Company CS7: This company produces automated warehouse solutions. The variety of warehouses and requirement differences are addressed with configurability for selected functionalities. The software is configured with a custom-developed configurator tool. They are testing pre-programmed scenarios on specific configurations, which are selected by experts. However, the workshop participant stated that currently, it is not possible to test many configurations from their large configuration space.

Company CS8: The last company in our study is providing automation solutions for a wide range of industry domains. They are operating with a diverse set of hardware, which entails a high degree of variability. Moreover, their solutions are deployed in different domains which motivates adaptability to different circumstances and requirements. This variability is documented in different lists with different levels of structure. They use a variety of techniques for testing different configurations. Some tests are specific for a single configuration, others can be run on multiple configurations, and other tests can be configured to match the tested configuration.

5.2. Results

Table 2 lists the participants of our interactive survey, with the company identifier, the workshop (*WS*) they were part of, their experience, and their roles within the company.

Table 2
Survey phase 2 participants.

Case	WS	Exp.	Role
CS1	1	> 20	developer, modeler, team leader, project manager, domain expert, researcher, product owner, system architect, software architect
CS2	2	5–10	developer
CS3	2	11–20	developer
CS4	3	11–20	developer, team leader, project manager, domain expert, product owner, system owner, system architect
CS5	1	> 20	team leader, domain expert, product owner, system architect
CS6	3	11–20	developer, project manager, researcher, product owner, system owner, system architect, software architect
CS7	3	3–5	developer, team leader, software architect
CS8	3	11–20	team leader, domain expert, QA manager

5.2.1. RQ2-1: Variability management

Fig. 4 displays the answers to how variability is specified and implemented in the participating companies. Although knowledge of variability management concepts and software product lines is widespread (see next subsections), currently, only CS5 is investigating the use of a variability model for some parts of their platform. The most common way variability is being documented at the studied companies is in lists and spreadsheets describing configuration options. Two companies expressed interest in introducing a variability model in the future. However, many of the participants voiced concerns about the effort of introducing (and maintaining) variability models, especially for their legacy systems. As the representative from CS4 stated: *The effort to create a model of all the parameters/options available is high, and the resources required for this are basically not available.* CS8 uses different tools for different purposes in different departments. They use an ERP tool for production-related variability. For sales, they use spreadsheets to find the right variant for a customer. Furthermore, for communicating the configuration to production they use simple text documents.

All eight companies answered that the variability documentation is maintained in the software/system development department. *Only CS1 has a dedicated platform team* that is responsible for variability documentation of the entire system. *Different departments have distinct views on variability* that are important for the particular department and which have to be consolidated across departments. Two companies (CS4, CS8) have a product management department that is also responsible for managing the variability of the entire system. More research to help consolidate those views and incorporate existing variability documentations is required.

The most common variability mechanism is *conditional execution (i.e., option/feature toggles)* with configuration options in files or databases. *Most companies use a variety of variability mechanisms.*

We further asked our survey participants what challenges they are facing with their current variability mechanisms (Fig. 5). The most common answer was *identifying variants affected by a change*. For instance, a bug was discovered and fixed and the change has to be rolled out to the customers that are affected by the bug. Related to this, five of the participants also answered

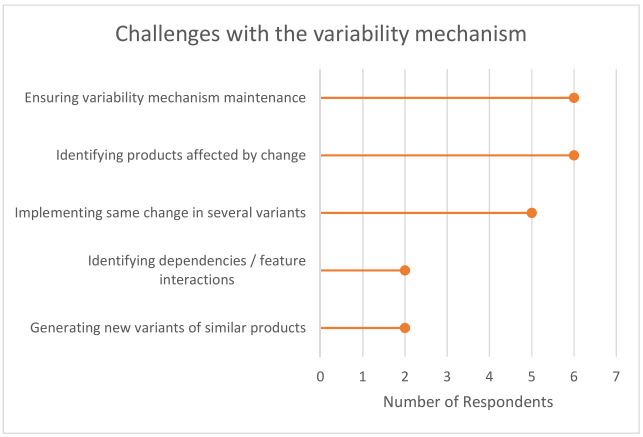


Fig. 5. Challenges with the variability mechanism.

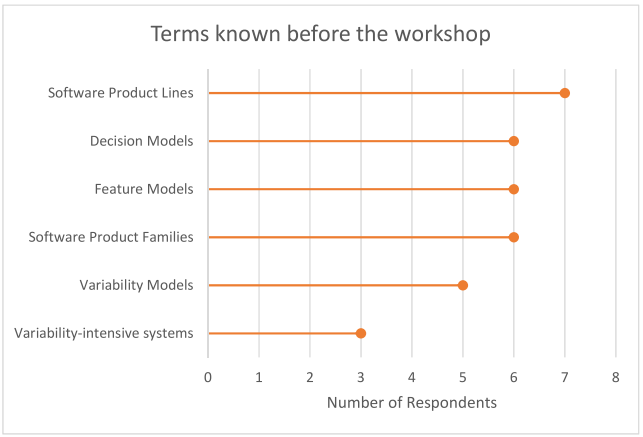


Fig. 6. Terms known before the workshop.

that implementing the same change in multiple variants is often a challenge for them. Moreover, the majority of participants reported that ensuring the correct maintenance of the variants was challenging. Finally, CS7 and CS8 both mentioned that they are confronted with challenges regarding the identification of variability dependencies. According to our participant from CS8 this is especially an issue with dependencies across hard- and software. The variability in these different areas is often developed within separate departments, without a clear link to how it manifests beyond their specialized view over the entire system. For instance, it might not be clear what effect changing the configuration of the software has on the hardware configuration that might also require changes. Improved solutions of tracing feature implementations and their interactions are required to alleviate these issues.

The configuration of the systems is generally done by custom-developed configuration tools that include hard-coded variability knowledge. These proprietary tools range from a custom domain-specific language used to build a configuration (CS1) to applications that enable the selection of a configuration, which can then be automatically built for deployment (CS2, CS3, CS4, CS6, CS7). Only CS5 is investigating the use of a variability model. To reduce the creation of numerous proprietary configuration tools the development of an extendable configurator supporting various variability documentation formats and mechanisms could be useful.

Moreover, we were interested in which terms commonly used in research were known to our participants before our workshops. Fig. 6 shows the terms that our participants answered as knowing before the workshop. The majority of the participants were familiar with most if the terms. Only *Variability-intensive systems* was known by merely three participants. Moreover, there were some differences in which types of models the participants had heard of before. Some only knew decision models while others only knew feature models. Only two participants were familiar with all three types of models. This might also explain why most of the companies develop their own configuration tools and do not utilize existing solutions based on common model types.

5.2.2. RQ2-2: Testing highly configurable CPSs

Fig. 7 shows our survey results regarding the testing of the configurable systems. While 4 of the participants answered that the software configuration is automated (Fig. 7(c)), only CS8 designs tests to be configurable as parametrized tests within their test automation framework (cf. RQ1-1 and RQ1-3a). Many tests are developed specifically for a configuration or a subset of the configurations, most commonly by employing clone-and-own (cf. RQ1-1) as depicted in Fig. 7(a). Existing test frameworks often are lacking support for configuring test cases to test different system variants.

Moreover, there is no systematic testing of all possible configurations. Generally, tested configurations and tests are selected by an expert (cf. RQ1-2) and no automated selection is used (Fig. 7(b)), as proposed in research (Lopez-Herrejon et al., 2015). CS2 mentioned that the lack of knowledge about which configurations are used in practice is a challenge for selecting configurations to test (cf. RQ1-2) by the development departments. Applying improved solutions to document variability and track existing configurations could be a useful avenue for future research.

Finally, independently from our survey, some companies mentioned issues with supply chain shortages due to the COVID-19 pandemic. This affected variability because they had to switch to different hardware components, simply because these were the only available ones. The participant from CS8 mentioned challenges in sufficiently testing these newly emerging configurations (cf. RQ1-3b). Utilizing research solutions to document variability and dependencies could help to better deal with such unforeseeable changes and to automatically select (cf. RQ1-2) the configurations to test with the new hardware.

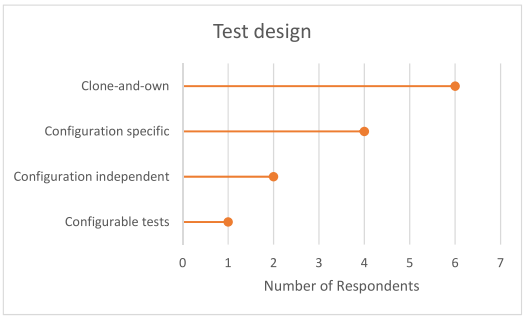
5.3. Threats to validity

Like the previous phase, phase 2 faces some threads to validity.

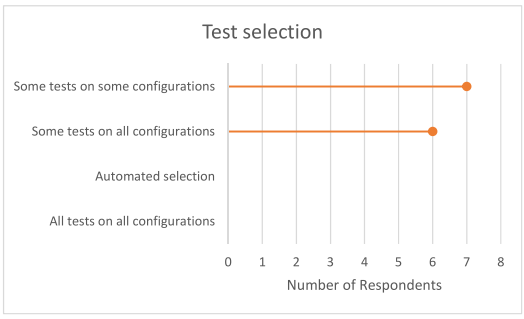
Construct Validity. Common threats to the validity of such empirical studies are misunderstanding questions by participants or misunderstanding participants' answers by the researchers. To mitigate this, we presented the basics of variability management and testing at the start of the workshops in order to establish a common understanding of the relevant concepts and terms. Moreover, we encouraged participants to ask questions while completing the questionnaire, especially if they were unsure about any of the questions or response options.

Internal Validity. An additional threat is the selection of participants. The participants in the survey were selected by the authors because they are contacts from the companies participating in ongoing or previous collaborations.

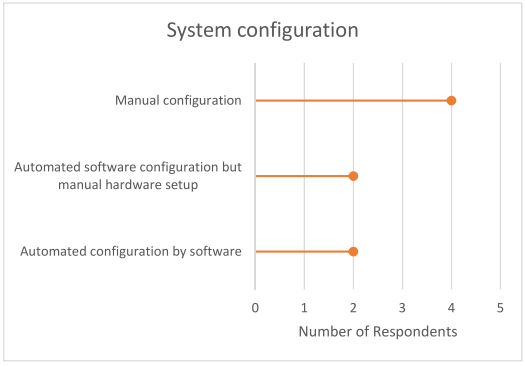
External Validity. We included four additional companies in phase 2. Like the four companies of the first phase, these are international companies which have offices in different parts of the world, they sell their products globally, and the development is done by international teams. Nonetheless, the representatives participating in our survey were again all located in Austria, which might have introduced some cultural biases.



(a) Test design.



(b) Test selection.



(c) System configuration.

Fig. 7. Test strategies.

6. Lessons learned

We were able to identify common issues and challenges reported by the participating companies in both phases of our study. In this section, we discuss these findings in terms of lessons learned (LL) and their implications for practitioners and researchers together with some ideas for future research.

LL1. Invest in variability early and know why. In both phases we found that variability models are not used in industry, but custom-developed configurators or spreadsheet-based approaches are applied to manage variability. Only one of the eight companies that participated in our study is in the process of introducing variability models for their control software, working together with academic researchers. We inquired about the reasons why the companies did not adopt variability models and software product line approaches. The most common rationale is that variability increases slowly over time (more and more variants are created for customers) and therefore companies felt early on that no variability model was required. However, over time, the systems grow and issues with variability start to arise. Then a variability model and a product line approach would become worthwhile, but the variability of the system has already reached a level of complexity that makes introducing a variability model increasingly difficult and costly. Moreover, the discussion with the practitioners revealed that not all benefits of having a variability model were immediately obvious to them. While many benefits are clearly described in the software product line literature (Martinez et al., 2017), this knowledge did at least not reach the participants of our studies. Furthermore, some companies simply did not know about variability modeling. Researchers aiming to collaborate with industry should focus on clearly motivating the potential benefits (and risks) of introducing variability models, e.g., by referring to existing empirical studies (Martinez et al., 2017). The research community as a whole could aim to increase research transfer activities, where possible in collaboration with tool vendors (Beuche, 2016; Krueger and Clements, 2018). Practitioners aware of the benefits of variability modeling should invest in systematic variability management as soon as possible. Refactoring legacy systems into a product line is possible (Assunção et al., 2017) and a core focus of the ongoing research.

LL2. Automation is necessary, but not a silver bullet for creating and maintaining variability models. Even if investing the effort to create variability models for an existing complex, highly configurable CPS, the question remains how to maintain and evolve these models over time and keep them consistent with the implementation. Many participants answered that adopting and maintaining a variability model is just too expensive. For researchers, this issue can be a motivation to work on new approaches for automating the generation of variability models (She et al., 2014). Existing approaches face challenges of requiring knowledge of valid existing variants, which in practice are not always available in the necessary detail and quantity to synthesize an accurate variability model. Another challenge is that much of the variability knowledge might be contained in domain-specific artifacts that require individual solutions to analyze and extract (Assunção et al., 2017). Especially in CPSs, artifacts from diverse engineering disciplines, created with diverse tools need to be analyzed. Moreover, maintaining a variability model and ensuring it stays consistent with the variability in diverse system artifacts can be challenging for practitioners. Research efforts to automatically update a variability model from other changes could be beneficial to mitigate this challenge (Feichtinger et al., 2021a). It may be a worthwhile research avenue to adopt approaches from other domains for automated consistency checking and change propagation for variability model changes (Pandolfo et al., 2021; Kretschmer et al., 2021). However, despite automation being necessary and helpful to create and maintain variability models, systematic round-trip/evolution processes allowing human intervention will still be required.

LL3. Regarding variability, never limit yourself to just one point of view. Another challenge for dealing with variability in a cyber–physical industrial system is the distribution of variability knowledge over different departments and engineering disciplines within a company. These departments and disciplines often have different views of the variability of the overall system. For example, some options might only be relevant for the software system, other options are related to mechanical components. A variability model for the entire system could help and consolidate these different views and ensure they do not diverge over time. For such a model a dedicated team is needed that collaborates with all affected departments. This, however, is often not feasible in practice, especially in project-driven businesses. Researchers have thus proposed different approaches to manage variability in multiple sub-models that enable each department to have and keep its own individual model, which includes links and constraints to other sub-models (Fischer et al., 2015; Schröter et al., 2016; Fadhlillah et al., 2022). Some approaches even support diverse model types (Galindo et al., 2015; Fadhlillah et al., 2022). These approaches still need to be validated with industrial-size case studies (Rabiser et al., 2018).

LL4. Watch out for diamonds in the rough: improve transfer from research to industry when it comes to variability and testing. There exist standards for engineering configurable systems (IEC, 2021). However, for variability modeling standardization is missing, despite previous and ongoing efforts (Haugen et al., 2013; Sundermann et al., 2021). This further complicates technology transfer from research to industry as many approaches from research presuppose the availability of a variability model. For instance, Combinatorial Interaction Testing (CIT) and similar sampling techniques use a variability model to select configurations for testing (Lopez-Herrejon et al., 2015). While the goal should not be to replace existing approaches with just one standard, unifying existing research (Haugen et al., 2013; Sundermann et al., 2021), e.g., by providing means to transform (Feichtinger et al., 2021b) or exchange (Schulze and Hellebrand, 2015) existing models could help to bring more potentially useful approaches from research to industry. While many research approaches and tools might not be directly applicable to real, large-scale industrial systems, there still might be some diamonds in the rough.

LL5. Reuse to reduce testing effort. In both study phases we found that tests are very rarely reused for different configurations. Rather tests are commonly implemented for specific configurations. In cases where tests are reused, it is done mainly by manual clone-and-own: tests for one configuration are copied and adapted to work on a different configuration. Automated reuse of tests and test artifacts for different configurations (Reuys et al., 2006; Fischer et al., 2019, 2020) or generating and configuring tests (Kästner et al., 2012; Fischer et al., 2021b) are the exception. Only one of the eight company partners in phase 2 reported having a few tests that can be configured automatically. Moreover, none of the company partners use any automated support like CIT for systematically covering configuration options testing. Instead, domain experts select the configurations to be tested based on their experience. Pending challenges like the lack of a standardized variability model (see above) and scalability issues of the sampling approaches (Pett et al., 2019) seem to hamper adoption of such techniques in practice.

LL6. Beware of the … "physical" in a CPS. Finally, testing CPSs commonly requires complex test environments, either allowing testing on the real hardware or with simulations up to a complete digital twin Dalibor et al. (2022) of the system. Variability further complicates this approach, because not only the software needs to be configured but also the hardware configuration (or the digital twin) might change. Therefore, either different hardware has to be available for testing or the used simulations also have to support different hardware configurations. Dependencies on real hardware are a major limiting factor for testing a wide range of configurations, especially if physical devices or hardware extensions are required to setup these configurations. Providing and maintaining a large collection of hardware devices for testing is often considered infeasible or simply too expensive in practice. Most of the systems of our partner companies require some manual configuration and the system cannot automatically be reconfigured to test multiple configurations. This issue further restricts the number of configurations that can be tested due to the involved manual setup effort. Furthermore, practitioners reported that some tests require manual interaction with the test system and test execution can only be automated partially. Without automation, however, testing a large number of configurations is practically impossible. Thus, research on testing product lines (do Carmo Machado et al., 2014b; Lee et al., 2020) should be combined with the development of strategies for automated testing of CPS (Ramler et al., 2014) to support applications in industry.

7. Conclusion

In this paper, we presented a two-phased multiple case study on the state of the practice of testing highly configurable CPSs. In phase 1 we first performed a survey with subsequent interviews with four companies. In phase 2, we held interactive workshops with the four original case study partners and with four additional companies. Therefore, we were able to study in detail industry cases from eight different domains, encompassing a variety of techniques and requirements. The aim of the study was to gain an understanding of common challenges and possible solutions regarding variability and testing in the context of CPSs. In summary, we found rather pragmatic approaches applied in the involved case companies that have developed over time. For managing variability, for example, the companies commonly created their own solutions with custom tools, models, and processes. Variability models, as proposed in software product line research, are mostly unknown and are not used. Some of the practitioners were skeptical about the usefulness and effort of introducing and maintaining variability models. Considering the experiences from the software product line community, we are confident that a variability model can help to overcome some of the challenges our case study companies are dealing with. Moreover, utilizing approaches from research supporting testing configurable systems often requires using a variability model, e.g., approaches for selecting configurations that should be tested. In our partner companies, this selection is currently done manually by domain experts based on many years of experience. Additionally, in practice, very few test cases are implemented to be configurable. Instead, tests are developed for specific configurations and these tests are then manually reused for other configurations following a clone-and-own reuse approach. Dependencies on complex testing environments in CPSs often involve using hardware devices that have to be configured manually, which limits the degree of automation in testing. The lack of test automation in turn limits the feasibility of testing a large number of system configurations.

In future work it would be interesting to repeat the study with international companies developing and maintaining CPSs. We plan to partner with international research teams for this purpose. Also, guidelines could be developed to support companies in automating their testing practices, facilitating reuse of test artifacts, and for developing configurable test cases. The research community should keep on improving approaches to automatically populate and maintain variability models and particularly, also support custom-developed artifacts, engineering models, and domain-specific (programming) languages as sources of variability information. Furthermore, unifying existing variability modeling approaches is important for industrial adoption as well as for improving the support for multiple disciplines in CPSs.

CRediT authorship contribution statement

Stefan Fischer: Methodology, Validation, Investigation, Data curation, Writing – original draft, Writing – review & editing. **Claus Klammer:** Methodology, Validation, Investigation, Resources, Data curation, Writing – original draft. **Antonio Manuel Gutiérrez Fernández:** Methodology, Investigation, Writing - original draft, Writing – review & editing, Visualization. **Rick Rabiser:** Methodology, Investigation, Resources, Writing – original draft, Writing – review & editing, Project administration. **Rudolf Ramler:** Conceptualization, Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

The research reported in this paper has been funded by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK), Austria, the Federal Ministry for Digital and Economic Affairs (BMDW), Austria, and the State of Upper Austria in the frame of the COMET — Competence Centers for Excellent Technologies Program managed by Austrian Research Promotion Agency FFG, Austria. The financial support by the Christian Doppler Research Association, Austria, the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development, Austria is gratefully acknowledged.

References

Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P., 2012. On extracting feature models from product descriptions. In: Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25-27, 2012. Proceedings. pp. 45–54.

Ahmed, F., Capretz, L.F., 2015. A framework for process assessment of software product line. CoRR abs/1507.06948, arXiv:1507.06948.

Assunção, W.K., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A., 2017. Reengineering legacy applications into software product lines: A systematic mapping. Empir. Softw. Eng. 22 (6), 2972–3016.

Bashroush, R., Garba, M., Rabiser, R., Groher, I., Botterweck, G., 2017. CASE tool support for variability management in software product lines. ACM Comput. Surv. 50 (1), 14:1–14:45.

Berger, T., Lettner, D., Rubin, J., Grünbacher, P., Silva, A., Becker, M., Chechik, M., Czarnecki, K., 2015. What is a feature?: A qualitative study of features in industrial software product lines. In: Proceedings of the 19th International Conference on Software Product Line. SPLC 2015, Nashville, TN, USA, July 20-24, 2015, pp. 16–25.

Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wasowski, A., 2013a. A survey of variability modeling in industrial practice. In: The Seventh International Workshop on Variability Modelling of Software-Intensive Systems. VaMoS '13, Pisa , Italy, January 23 - 25, 2013, pp. 7:1–7:8.

Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K., 2013b. A study of variability models and languages in the systems software domain. IEEE Trans. Softw. Eng. 39 (12), 1611–1640.

Berger, T., Steghöfer, J.-P., Ziadi, T., Robin, J., Martinez, J., 2020. The state of adoption and the challenges of systematic variability management in industry. Empir. Softw. Eng. 25.

Beuche, D., 2016. Using pure: Variants across the product line lifecycle. In: Proceedings of the 20th International Systems and Software Product Line Conference. pp. 333–336.

Biffl, S., Lüder, A., Gerhard, D. (Eds.), 2017. Multi-Disciplinary Engineering for Cyber-Physical Production Systems, Data Models and Software Solutions for Handling Complex Engineering Projects. Springer, http://dx.doi.org/10.1007/978-3-319-56345-9.

Biffl, S., Mätzler, E., Wimmer, M., Lüder, A., Schmidt, N., 2015. Linking and versioning support for AutomationML: A model-driven engineering perspective. In: 13th IEEE International Conference on Industrial Informatics. INDIN 2015, Cambridge, United Kingdom, July 22-24, 2015, IEEE, pp. 499–506.

Corbin, J., Strauss, A., 2008. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, third ed. Sage Publications, Inc..

Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wasowski, A., 2012. Cool features and tough decisions: A comparison of variability modeling approaches. In: Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems. VaMoS '12, Association for Computing Machinery, New York, NY, USA, pp. 173–182.

Dalibor, M., Jansen, N., Rumpe, B., Schmalzing, D., Wachtmeister, L., Wimmer, M., Wortmann, A., 2022. A cross-domain systematic mapping study on software engineering for digital twins. J. Syst. Softw. 111361.

do Carmo Machado, I., McGregor, J.D., Cavalcanti, Y.C., de Almeida, E.S., 2014a. On strategies for testing software product lines: A systematic literature review. Inf. Softw. Technol. 56 (10), 1183–1199.

do Carmo Machado, I., McGregor, J.D., Cavalcanti, Y.C., De Almeida, E.S., 2014b. On strategies for testing software product lines: A systematic literature review. Inf. Softw. Technol. 56 (10), 1183–1199.

Dubinsky, Y., Rubin, J., Berger, T., Duszynski, S., Becker, M., Czarnecki, K., 2013. An exploratory study of cloning in industrial software product lines. In: 17th European Conference on Software Maintenance and Reengineering. CSMR 2013, Genova, Italy, March 5-8, 2013, pp. 25–34.

Eckert, K., Fay, A., Hadlich, T., Diedrich, C., Frank, T., Vogel-Heuser, B., 2012. Design patterns for distributed automation systems with consideration of non-functional requirements. In: Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation. ETFA 2012, pp. 1–9.

Engström, E., Runeson, P., 2010. A qualitative survey of regression testing practices. In: Product-Focused Software Process Improvement, 11th International Conference, PROFES 2010, Limerick, Ireland, June 21-23, 2010. Proceedings. pp. 3–16.

Engström, E., Runeson, P., 2011. Software product line testing - A systematic mapping study. Inf. Softw. Technol. 53 (1), 2–13.

Fadhlillah, H.S., Feichtinger, K., Meixner, K., Sonnleithner, L., Rabiser, R., Zoitl, A., 2022. Towards Multidisciplinary Delta-oriented variability management in cyber-physical production systems. In: Arcaini, P., Devroey, X., Fantechi, A. (Eds.), VaMoS '22: 16th International Working Conference on Variability Modelling of Software-Intensive Systems. Florence, Italy, February 23 - 25, 2022, ACM, pp. 13:1–13:10.

Fay, A., Vogel-Heuser, B., Frank, T., Eckert, K., Hadlich, T., Diedrich, C., 2015. Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns. J. Syst. Softw. 101, 221–235.

Feichtinger, K., Hinterreiter, D., Linsbauer, L., Prähofer, H., Grünbacher, P., 2021a. Guiding feature model evolution by lifting code-level dependencies. J. Comput. Lang. 63, 101034.

Feichtinger, K., Stöbich, J., Romano, D., Rabiser, R., 2021b. Travart: An approach for transforming variability models. In: 15th International Working Conference on Variability Modelling of Software-Intensive Systems. ACM, pp. 1–10.

Fischer, J., Bougouffa, S., Schlie, A., Schaefer, I., Vogel-Heuser, B., 2018. A qualitative study of variability management of control software for industrial automation systems. In: 2018 IEEE International Conference on Software Maintenance and Evolution. ICSME 2018, Madrid, Spain, September 23-29, 2018, pp. 615–624.

Fischer, S., Klammer, C., Gutierrez, A., Rabiser, R., Ramler, R., 2022. Online appendix. https://github.com/stefFischer/2022-Journal-Appendix-ConfigurableTestingIndustrySurvey.

Fischer, S., Linsbauer, L., Lopez-Herrejon, R.E., Egyed, A., Ramler, R., 2015. Bridging the gap between software variability and system variant management: Experiences from an industrial machinery product line. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications. IEEE, pp. 402–409.

Fischer, S., Michelon, G.K., Ramler, R., Linsbauer, L., Egyed, A., 2020. Automated test reuse for highly configurable software. Empir. Softw. Eng. 25 (6), 5295–5332.

Fischer, S., Ramler, R., Klammer, C., Rabiser, R., 2021a. Testing of highly configurable cyber-physical systems - A multiple case study. In: 15th International Working Conference on Variability Modelling of Software-Intensive Systems. VaMoS 2021, ACM, Krems, Austria, pp. 19:1–19:10.

Fischer, S., Ramler, R., Linsbauer, L., 2021b. Comparing automated reuse of scripted tests and model-based tests for configurable software. In: 2021 28th Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 421–430.

Fischer, S., Ramler, R., Linsbauer, L., Egyed, A., 2019. Automating test reuse for highly configurable software. In: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume a. pp. 1–11.

Galindo, J.A., Alférez, M., Acher, M., Baudry, B., Benavides, D., 2014. A variability-based testing approach for synthesizing video sequences. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis. In: ISSTA 2014, Association for Computing Machinery, New York, NY, USA, pp. 293–303.

Galindo, J.A., Dhungana, D., Rabiser, R., Benavides, D., Botterweck, G., Grünbacher, P., 2015. Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. Inf. Softw. Technol. 62, 78–100.

Galindo, J.A., Turner, H.A., Benavides, D., White, J., 2016. Testing variability-intensive systems using automated analysis: An application to android. Softw. Qual. J. 24 (2), 365–405.

Garousi, V., Afzal, W., Çaglar, A., Isik, I.B., Baydan, B., Çaylak, S., Boyraz, A.Z., Yolaçan, B., Herkiloglu, K., 2017. Comparing automated visual GUI testing tools: An industrial case study. In: Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing. a-TEST@ESEC/SIGSOFT FSE 2017, Paderborn, Germany, September 4-5, 2017, pp. 21–28.

Harrison, R., Vera, D.A., Ahmad, B., 2016. Engineering methods and tools for cyber-physical automation systems. Proc. IEEE 104 (5), 973–985.

Haugen, Ø., Wasowski, A., Czarnecki, K., 2013. CVL: Common variability language. In: Kishi, T., Jarzabek, S., Gnesi, S. (Eds.), 17th International Software Product Line Conference. SPLC 2013, Tokyo, Japan - August 26 - 30, 2013, ACM, p. 277.

Hierons, R.M., Li, M., Liu, X., Parejo, J.A., Segura, S., Yao, X., 2020. Many-objective test suite generation for software product lines. ACM Trans. Softw. Eng. Methodol. 29 (1).

Humeniuk, D., Antoniol, G., Khomh, F., 2021. Data driven testing of cyber physical systems. arXiv:2102.11491.

IEC, 2010. ISO/IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems. Part 1: General requirements. URL https://www.iec.ch.

IEC, 2013. IEC 61131 - programmable controllers, part 3: Programming languages: Edition 3.0. URL https://www.iec.ch.

IEC, 2021. ISO/IEC 26580:2021 - Software and systems engineering — Methods and tools for the feature-based approach to software and systems product line engineering. URL https://www.iec.ch.

Kang, K.C., Cohen, S., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-oriented domain analysis (FODA) feasibility study.

Kassab, M., 2018. Testing practices of software in safety critical systems: Industrial survey. In: Proceedings of the 20th International Conference on Enterprise Information Systems, ICEIS 2018, Funchal, Madeira, Portugal, March 21-24, 2018, Volume 2. pp. 359–367.

Kassab, M., DeFranco, J.F., Laplante, P.A., 2017. Software testing: The state of the practice. IEEE Softw. 34 (5), 46–52.

Kästner, C., Von Rhein, A., Erdweg, S., Pusch, J., Apel, S., Rendel, T., Ostermann, K., 2012. Toward variability-aware testing. In: Proceedings of the 4th International Workshop on Feature-Oriented Software Development. pp. 1–8.

Kim, C.H.P., Batory, D.S., Khurshid, S., 2011. Reducing combinatorics in testing product lines. In: Proceedings of the 10th International Conference on Aspect-Oriented Software Development. AOSD 2011, Porto de Galinhas, Brazil, March 21-25, 2011, pp. 57–68.

Kretschmer, R., Khelladi, D.E., Lopez-Herrejon, R.E., Egyed, A., 2021. Consistent change propagation within models. Softw. Syst. Model. 20 (2), 539–555.

Krueger, C., Clements, P., 2018. Feature-based systems and software product line engineering with gears from BigLever. In: Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 2. pp. 1–4.

Lam, W., Wu, Z., Li, D., Wang, W., Zheng, H., Luo, H., Yan, P., Deng, Y., Xie, T., 2017. Record and replay for android: Are we there yet in industrial cases? In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017, pp. 854–859.

Lamancha, B.P., Polo, M., Piattini, M., 2010. Systematic review on software product line testing. In: Software and Data Technologies - 5th International Conference. ICSOFT 2010, Athens, Greece, July 22-24, 2010. Revised Selected Papers, pp. 58–71.

Lee, J., Kang, S., Jung, P., 2020. Test coverage criteria for software product line testing: Systematic literature review. Inf. Softw. Technol. 122, 106272.

Li, Z., Chen, J., Jiao, R., Zhao, N., Wang, Z., Zhang, S., Wu, Y., Jiang, L., Yan, L., Wang, Z., Chen, Z., Zhang, W., Nie, X., Sui, K., Pei, D., 2021. Practical root cause localization for microservice systems via trace analysis. In: 29th IEEE/ACM International Symposium on Quality of Service. IWQOS 2021, Tokyo, Japan, June 25-28, 2021, IEEE, pp. 1–10. http://dx.doi.org/10.1109/IWQOS52092.2021.9521340.

Lopez-Herrejon, R.E., Ferrer, J., Chicano, F., Egyed, A., Alba, E., 2014. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In: 2014 IEEE Congress on Evolutionary Computation. CEC, pp. 387–396.

Lopez-Herrejon, R.E., Fischer, S., Ramler, R., Egyed, A., 2015. A first systematic mapping study on combinatorial interaction testing for software product lines. In: Eighth IEEE International Conference on Software Testing, Verification and Validation, ICST 2015 Workshops. Graz, Austria, April 13-17, 2015, pp. 1–10.

Martinez, J., Assunção, W.K., Ziadi, T., 2017. ESPLA: A catalog of extractive SPL adoption case studies. In: Proceedings of the 21st International Systems and Software Product Line Conference-Volume B. pp. 38–41.

Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T., Saake, G., 2017. Mastering software variability with featureide. Springer.

Metzger, A., Pohl, K., 2014. Software product line engineering and variability management: Achievements and challenges. In: Future of Software Engineering Proceedings. FOSE 2014, Association for Computing Machinery, New York, NY, USA, pp. 70–84.

Mossige, M., Gotlieb, A., Spieker, H., Meling, H., Carlsson, M., 2017. Time-aware test case execution scheduling for cyber-physical systems. In: Beck, J.C. (Ed.), Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. In: Lecture Notes in Computer Science, vol. 10416, Springer, pp. 387–404. http://dx.doi.org/10.1007/978-3-319-66158-2_25.

Mukelabai, M., Nesic, D., Maro, S., Berger, T., Steghöfer, J., 2018. Tackling combinatorial explosion: A study of industrial needs and practices for analyzing highly configurable systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. ASE 2018, Montpellier, France, September 3-7, 2018, pp. 155–166.

Nguyen, H.V., Kästner, C., Nguyen, T.N., 2014. Exploring variability-aware execution for testing plugin-based web applications. In: 36th International Conference on Software Engineering. ICSE '14, Hyderabad, India - May 31 - June 07, 2014, pp. 907–918.

Pandolfo, L., Pulina, L., Vuotto, S., 2021. SMT-based consistency checking of configuration-based components specifications. IEEE Access 9, 83718–83726.

Parejo, J.A., Sánchez, A.B., Segura, S., Cortés, A.R., Lopez-Herrejon, R.E., Egyed, A., 2016. Multi-objective test case prioritization in highly configurable systems: A case study. J. Syst. Softw. 122, 287–310.

Patel, S., Shah, V., 2015. Automated testing of software-as-a-service configurations using a variability language. In: Schmidt, D.C. (Ed.), Proceedings of the 19th International Conference on Software Product Line. SPLC 2015, Nashville, TN, USA, July 20-24, 2015, ACM, pp. 253–262. http://dx.doi.org/10.1145/2791060.2791072.

Pett, T., Thüm, T., Runge, T., Krieter, S., Lochau, M., Schaefer, I., 2019. Product sampling for product lines: The scalability challenge. In: Proceedings of the 23rd International Systems and Software Product Line Conference. SPLC 2019, Volume a, Paris, France, September 9-13, 2019, pp. 14:1–14:6.

Raatikainen, M., Tiihonen, J., Männistö, T., 2019. Software product lines and variability modeling: A tertiary study. J. Syst. Softw. 149, 485–510.

Rabiser, R., Schmid, K., Becker, M., Botterweck, G., Galster, M., Groher, I., Weyns, D., 2018. A study and comparison of industrial vs. Academic software product line research published at SPLC. In: 22nd International Systems and Software Product Line Conference. SPLC 2018, ACM, pp. 14–24.

Rafique, O., Schneider, K., 2020. Employing OpenCL as a standard hardware abstraction in a distributed embedded system: A case study. In: 9th Mediterranean Conference on Embedded Computing. MECO 2020, Budva, Montenegro, June 8-11, 2020, IEEE, pp. 1–7. http://dx.doi.org/10.1109/MECO49872.2020.9134270.

Ramler, R., Putschögl, W., Winkler, D., 2014. Automated testing of industrial automation software: Practical receipts and lessons learned. In: Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation. pp. 7–16.

Reuys, A., Reis, S., Kamsties, E., Pohl, K., 2006. The scented method for testing software product lines. In: Software Product Lines. Springer, pp. 479–520.

Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131–164.

Schmid, K., Rabiser, R., Grünbacher, P., 2011. A comparison of decision modeling approaches in product lines. In: Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems. VaMoS '11, Association for Computing Machinery, New York, NY, USA, pp. 119–126.

Schröter, R., Krieter, S., Thüm, T., Benduhn, F., Saake, G., 2016. Feature-model interfaces: the highway to compositional analyses of highly-configurable systems. In: Dillon, L.K., Visser, W., Williams, L.A. (Eds.), Proceedings of the 38th International Conference on Software Engineering. ICSE 2016, Austin, TX, USA, May 14-22, 2016, ACM, pp. 667–678.

Schulze, M., Hellebrand, R., 2015. Variability exchange language-a generic exchange format for variability data.. In: Software Engineering (Workshops). pp. 71–80.

Sen, K., Kalasapur, S., Brutch, T.G., Gibbs, S., 2013. Jalangi: A tool framework for concolic testing, selective record-replay, and dynamic analysis of JavaScript. In: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013, pp. 615–618.

She, S., Ryssel, U., Andersen, N., Wasowski, A., Czarnecki, K., 2014. Efficient synthesis of feature models. Inf. Softw. Technol. 56 (9), 1122–1143.
Sinha, R., Patil, S., Gomes, L., Vyatkin, V., 2019. A survey of static formal methods for building dependable industrial automation systems. IEEE Trans. Ind. Inform. 15 (7), 3772–3783.
Sogeti, 2009. TPI Next - Business Driven Test Process Improvement, first ed. UTN Publishers.
Srikanth, H., Cohen, M.B., Qu, X., 2009. Reducing field failures in system configurable software: Cost-based prioritization. In: 2009 20th International Symposium on Software Reliability Engineering. pp. 61–70.
Sundermann, C., Feichtinger, K., Engelhardt, D., Rabiser, R., Thüm, T., 2021. Yet another textual variability language?: A community effort towards a unified language. In: Mousavi, M., Schobbens, P. (Eds.), SPLC '21: 25th ACM International Systems and Software Product Line Conference, Leicester, United Kingdom, September 6-11, 2021, Volume A. ACM, pp. 136–147.
Thummalapenta, S., de Halleux, J., Tillmann, N., Wadsworth, S., 2010. DyGen: Automatic generation of high-coverage tests via mining gigabytes of dynamic traces. In: Fraser, G., Gargantini, A. (Eds.), Tests and Proofs - 4th International Conference, TAP@TOOLS 2010, MáLaga, Spain, July 1-2, 2010. Proceedings. In: Lecture Notes in Computer Science, vol. 6143, Springer, pp. 77–93. http://dx.doi.org/10.1007/978-3-642-13977-2_8.
Tzoref-Brill, R., Maoz, S., 2018. Modify, enhance, select: Cco-evolution of combinatorial models and test plans. In: Leavens, G.T., Garcia, A., Pasareanu, C.S. (Eds.), Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018, ACM, pp. 235–245. http://dx.doi.org/10.1145/3236024.3236067.
Vierhauser, M., Rabiser, R., Grünbacher, P., 2014. A case study on testing, commissioning, and operation of very-large-scale software systems. In: 36th International Conference on Software Engineering, ICSE '14, Companion Proceedings. Hyderabad, India, May 31 - June 07, 2014, pp. 125–134.
Villela, K., Silva, A., Vale, T., de Almeida, E.S., 2014. A survey on software variability management approaches. In: 18th International Software Product Line Conference. SPLC '14, Florence, Italy, September 15-19, 2014, pp. 147–156.
Vogel-Heuser, B., Fay, A., Schaefer, I., Tichy, M., 2015. Evolution of software in automated production systems: Challenges and research directions. J. Syst. Softw. 110, 54–84.
Vogel-Heuser, B., Fischer, J., Hess, D., Neumann, E.M., Wurr, M., 2021. Managing variability and reuse of extra-functional control software in CPPS. In: 2021 Design, Automation Test in Europe Conference Exhibition, Vol. 2021-February. DATE, pp. 755–760.
Vogt, H., 2017. Designing test environments for cyber-physical systems. In: Mitschang, B., Nicklas, D., Leymann, F., Schöning, H., Herschel, M., Teubner, J., Härder, T., Kopp, O., Wieland, M. (Eds.), Datenbanksysteme Für Business, Technologie Und Web. BTW 2017, Gesellschaft für Informatik, Bonn, pp. 573–574.
Wang, W., Niu, N., Alenazi, M., Xu, L.D., 2019. In-place traceability for automated production systems: A survey of PLC and sysml tools. IEEE Trans. Ind. Informatics 15 (6), 3155–3162. http://dx.doi.org/10.1109/TII.2018.2878782.

Stefan Fischer received the M.Sc. and Doctoral degree in software engineering and computer science from Johannes Kepler University Linz, Linz, Austria. He is a Senior Researcher at the Software Competence Center Hagenberg Gmbh (SCCH), Hagenberg, Austria where he is responsible for research in the area of software testing and quality assurance. He has several years of experience in software engineering research and technology transfer. His main research interests include configuration-aware software testing, software analytics, and run-time monitoring.

DI(FH) Claus Klammer is senior research project manager at the Software Analytics and Evolution Group at the Software Competence Center Hagenberg Gmbh (SCCH), Austria. He studied at the University of Applied Sciences Upper Austria, School of Informatics/ Communications/Media, Hagenberg and received his degree in 2001. Since 2004 he works at SCCH and has more than 15 years of experience in industrial research projects. His current research interests include applied research and knowledge transfer in the area of software engineering, software testing and automated software quality control.

Antonio M. Gutiérrez is a post-doctoral researcher at the Christian Doppler Laboratory for Mastering Variability in Software-intensive Cyber–Physical Production Systems at the LIT Cyber–Physical Systems Lab at Johannes Kepler University Linz, Austria. He holds a Master's and a Ph.D. degree in Software Engineering from the University of Sevilla. He worked as engineer for several companies before joining academia. His research interests include but are not limited to software variability, service-oriented computing and business processes.

Rick Rabiser is full university professor for Software Engineering in Cyber–Physical Systems at the Linz Institute of Technology (LIT) and head of the LIT Cyber–Physical Systems Lab as well as head of the Christian Doppler Laboratory for Mastering Variability in Software-intensive Cyber–Physical Production Systems at Johannes Kepler University (JKU) Linz, Austria. He holds a Master's and a Ph.D. degree in Business Informatics as well as the habilitation in Practical Computer Science from JKU. His research interests include but are not limited to variability management, systems and software product lines, software evolution, automated software engineering, and usability of software engineering tools.

Rudolf Ramler is a research manager at Software Competence Center Hagenberg (SCCH), Austria. Rudolf holds a M.Sc. in Business Informatics from Johannes Kepler University Linz. He has more than 20 years of experience in applied research in the fields of software engineering, software quality assurance and testing, software analytics, and application lifecycle management. He is author of over 100 reviewed publications related to these topics, co-organizer and chair of international conferences and workshops, an ISTQB certified tester, and an IEEE and ACM member. His mission and passion are to support industry in turning research results into practically successful solutions.