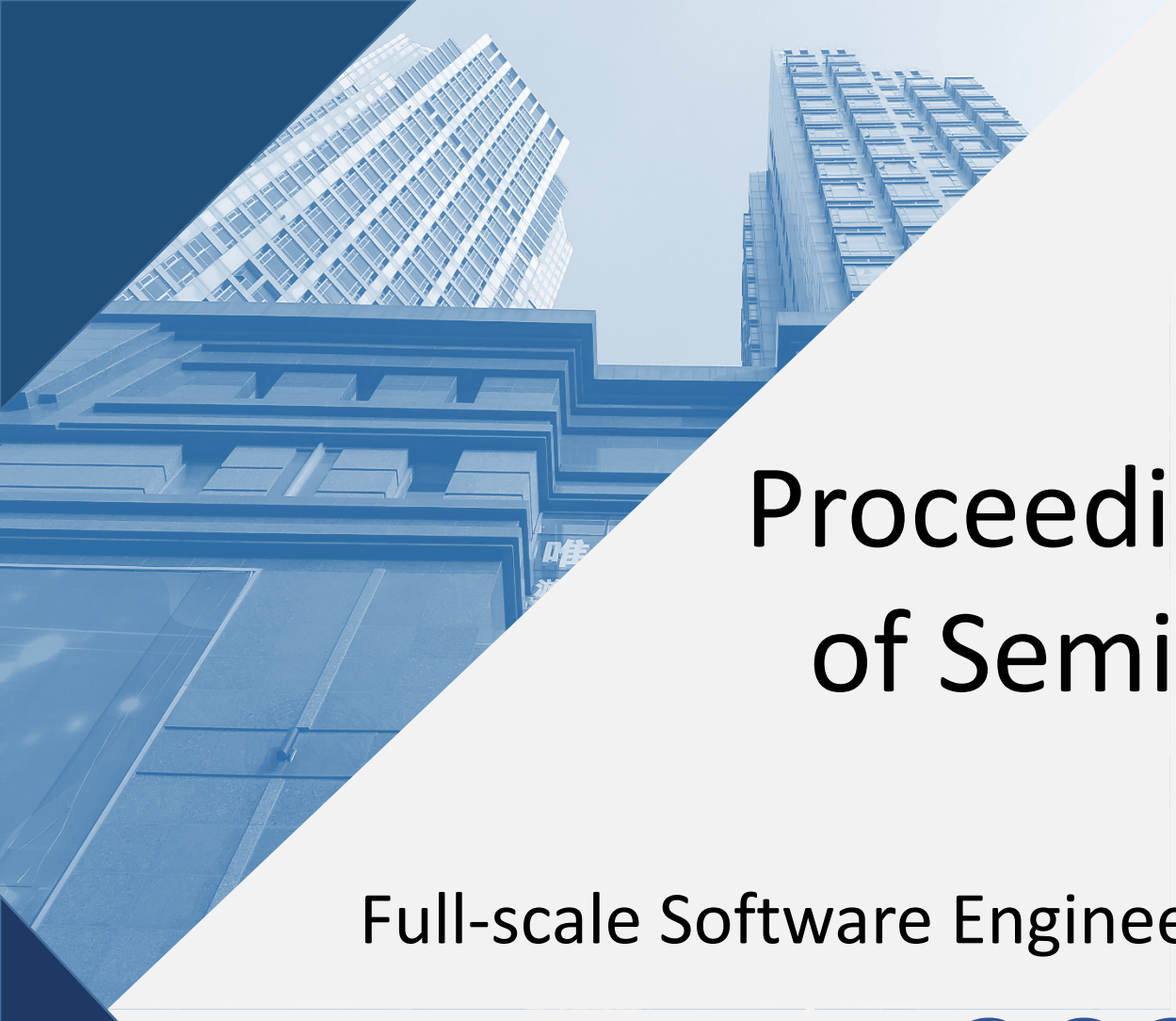

Proceedings of Seminar

Full-scale Software Engineering

2024

Editors: Horst Lichter
Nils Wild
Selin Aydin
Alex Sabau
Ada Slupczynski

SWC Software Construction | RWTH AACHEN UNIVERSITY

Table of Contents

Combinatorial Testing in Industry - State of Practice

A Systematic Literature Review

Nils Pulte
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
nils.pulte@rwth-aachen.de

Umer Amin Saleemi
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
umer.amin.saleemi@rwth-aachen.de

ABSTRACT

Testing is an integral part of software engineering to ensure that software functions as expected. However, exhaustive testing is often costly and infeasible. Combinatorial Testing (CT) is employed in industry to increase coverage while reducing the number of test cases. This paper conducts a Systematic Literature Review (SLR) to explore how CT is applied across various industry sectors, identify the tools utilized, and summarize the benefits, limitations, and drawbacks reported. A total of 37 publications with relevant information were identified and analyzed.

CT finds widespread application across various domains, such as autonomous driving and data management systems. Numerous tools and approaches have been developed to facilitate the creation of CT test suites, along with the underlying combinatorial models used for test generation. Research papers emphasize a multitude of advantages of CT, especially regarding efficiency in producing test suites with high coverage and small size. Additionally, CT offers significant benefits in terms of defect detection and resource optimization. However, there is also a set of factors that make the application of CT impractical, including the upfront effort required for tool integration and the necessity of substantial domain knowledge.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.5 [**Testing and Debuggingg**]: Testing—*Combinatorial Testing*

Keywords

Combinatorial Testing, Industry, Systematic Literature Review

1. INTRODUCTION

"Exhaustive testing is infeasible" [1, 2, 3]. This holds true for many real-world applications due to the number of possible combinations of parameters and values that would have to be considered. CT is praised to offer a partial solution to this problem as statistically, only a small number of parameters are involved in failures, so it is enough to test the combinations of those parameters instead of all possible combinations [4].

This paper conducts a SLR to find out about the practical application of CT in industry. It examines which *industries* are using CT, the *benefits* and *problems* are highlighted and the *tools* used. First, Chapter 2 provides an introduction to CT. Then, Chapter 3 explains how the steps of an SLR were applied in the context of this topic. Finally, Chapter 4 presents the findings, while Chapter [**?**] discusses threats to validity for the findings. Lastly, Chapter 6 summarizes the results.

2. COMBINATORIAL TESTING

Empirical observations suggest that "most failures are induced by single factor faults or by the joint combinatorial effect (interaction) of two factors, with progressively fewer failures induced by interactions between three or more factors" [4]. This observation is summarized in the *interaction rule*, and means that a test coverage equivalent to exhaustive testing can be practically reached by testing all t-tuples of parameters, where t describes the *interaction strength* that can be relatively low according to the interaction rule [5]. If for example all errors of a system are likely triggered by a combination of four or less parameters, it is enough to test all 4-way (which includes 3-way, pairwise and single) parameter combinations as opposed to testing every possible combination exhaustively.

The focus on parameter combinations summarizes the idea of CT and highlights the benefit that the number of test cases can be drastically reduced compared to actual exhaustive testing, while still providing a sense of exhaustive testing. Furthermore, CT considers that there are combinations of parameters and values that are invalid in a system, for example because they would not occur together in the real world, and avoids the generation of test cases for those combinations, making the resulting suite more representative for the real relationship between parameters [4].

3. METHODOLOGY

This paper applies a SLR to conduct the research. An SLR is a methodology used to systematically gather and process information from research related to a given topic.

SWC Seminar 2024 RWTH Aachen University, Germany.

It aims to synthesize findings, provide an overview of the current state of research, and identify existing gaps. The key to an SLR is its systematic organization. This systematic approach yields the benefit that researcher biases are avoided: While biases from primary sources can still carry over to the results, the methodology ensures that for example not individual papers are selected based on the initial idea of the outcome of the researcher. Furthermore, the process is designed to be reproducible to a certain degree (e.g. query results may change over time) and make the process more transparent [6]. Individual steps can be independently conducted by multiple authors, with a process in place to resolve conflicting opinions, thereby reducing the influence of individual subjective biases. [7].

The tool Parsifal[1] is used to facilitate the SLR. This tool guides users through the steps of the SLR, helps organize these steps, and documents the process. It is specifically designed for SLRs in the contextext of Software Engineering [8]. Parsifal splits the process into two major steps, namely planning and conducting. In the planning, the objectives of the research are defined and the research questions are formulated. Furthermore, a set of search engines is chosen as sources from which papers will be selected using search queries. To narrow down the resulting list of papers inclusion and exclusion criteria are chosen, that determine which sources yielded by the queried search engine will be part of the research and which will not. To assess individual studies and exclude those that yield little value on closer inspection a quality assessment checklist is defined. Lastly, in the data extraction form, a set of attributes is picked that defines the data points that are obtained from the primary sources. This planning step defines a protocol for the SLR to guide through the conduction phase and also record the individual steps to ensure the traceability of the results. In the conducting phase, the plan is now applied, meaning that the search engines are queried, the resulting set of primary sources are filtered, assessed, and finally, the data is extracted and analyzed.

In the following, the steps of an SLR will be discussed in more detail to show how they were applied to the given topic [9].

3.1 Research Goal

This section describes in more detail what the overall goal of the research is to highlight why which research questions are chosen.

This research is driven to uncover the state of practice on how CT is used in various industries. This includes finding out more about which areas and field of industry CT is applied in, but also which tools and methodologies are employed and which limitations and benefits of using CT are highlighted.

3.1.1 Research Questions

The overall objectives break down into the following research questions:

- **RQ1** In which domains of the industry is CT used and are there any observable trends?
- **RQ2** What are the challenges of applying CT in industry?
- **RQ3** What are the benefits of applying CT in industry?
- **RQ4** What tools are used to apply CT in industry?

[1]Parsifal, www.parsif.al

Criterion	Keyword	Synonyms
Population	Industry	Field, Industrial
Intervention	Combinatorial Testing	Combinatorial Test
Outcome	Tool	Technique, Utility
Outcome	Application	Use, Applying
Outcome	Benefits	Advantage, Improvement
Outcome	Drawbacks	Disadvantage, Limitations
Outcome	Lessons	Learnings
Context	Industry	Field, Industrial

Table 1: Overview over PICOC criteria, keywords and synonyms

3.1.2 PICOC

The components of the research questions consist of keywords categorized according to the following criteria[6]:

- The **population** defines the people/ areas/ entities that are affected by the intervention.
- The **intervention** describes which methodology /technology/ tool is studied.
- The **comparison** is what the intervention is compared with. It can be a control group for example. In the context of this paper the intervention is not further specified as a comparison of the intervention is not the main scope of this research.
- The **outcome** includes the aspects of the findings that are relevant for the overall goal of the research.
- The **context** defines where the research takes place, e.g. there could be a pure academic interest or the industrial context is of interest as it is the case for this paper.

These criteria are used to formulate database queries to identify primary sources during the SLR. In order to formulate more general queries and gather documents that do not fit the exact criteria those keywords are also rephrased and synonyms are added that can be used interchangeably with the keywords [9]. An overview of the keywords used in the search queries is given in Table 1

3.2 Search Strategy

This chapter describes the approach to gather primary sources and to decide which of those should be included and excluded in the research.

3.2.1 Search Engine

The selection of search engines is crucial for obtaining a comprehensive and unbiased overview of the topic in question. Thus for this review the *ACM Digital Library*[2], *DBLP*[3],

[2]ACM Digital Library, portal.acm.org
[3]DBLP, dblp.org

and *IEEE Digital Library*[4] were searched as a collection of articles relevant for computer science, along with more general databases for scientific literature like *ISI Web of Science*[5], *Science Direct*[6] and *Scopus*[7].

3.2.2 Search Strings

The search string is the query used to search the database. It is made up of the keywords associated with the PICOC 3.1.2 criteria. Generally, the individual categories are combined using *AND*, and the keywords and synonyms within each category are combined using *OR*. This yields the following base search string:

("Industry" OR "Field" OR "Industrial") AND ("Combinatorial Testing" OR "Combinatorial Test") AND ("Application" OR "Applying" OR "Use" OR "Benefits" OR "Advantage" OR "Improvement" OR "Drawbacks" OR "Disadvantage" OR "Limitations" OR "Lessons" OR "Learnings" OR "Tool" OR "Utility" OR "Technique")

This base string already gives the structure of the search strings for all databases, however for some databases the syntax has to be adapted. DBLP for example would not return results when simply pasting this base string, as instead a *space* represents an *AND* and a | represents an *OR*. So this base string must be slightly adapted for some searches.

3.2.3 Inclusion and Exclusion Criteria

Because not all scientific papers found using the search query end up being relevant for the topic of this paper, a set of inclusion and exclusion criteria is explicitly defined to determine which papers are going to be considered in more detail and which papers are excluded from further consideration in later steps of the review. To apply the criteria the title of the paper and the abstract are considered first, if these properties do not help finding a clear conclusion the paper should be skimmed. It is suggested that the inclusion criteria are defined before the review is conducted, to prevent bias, but that they can be refined later on [9], also, they should be derived from the research questions and piloted first to get a sense for their reliability [6]. To identify papers that are relevant to the topic the following inclusion criteria were chosen:

- **IC1** The paper describes areas in which CT is applied in industry
- **IC2** The paper describes tools used to apply CT

On the other hand the following criteria were picked to excluded papers that are not considered to keep the amount of resulting primary sources manageable and relevant.

- **EC1** The paper deals with CT, but in theory and not in an applied industrial context
- **EC2** The paper does not deal with CT
- **EC3** The paper is more than 10 years old
- **EC4** The paper is not in English
- **EC5** The full text is not accessible

[4]IEEE Digital Library, ieeexplore.ieee.org
[5]ISI Web of Science, www.isiknowledge.com
[6]Science Direct, www.sciencedirect.com
[7]Scopus, www.scopus.com

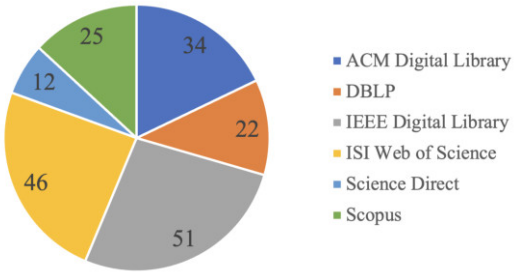

Figure 1: The number of papers found per source

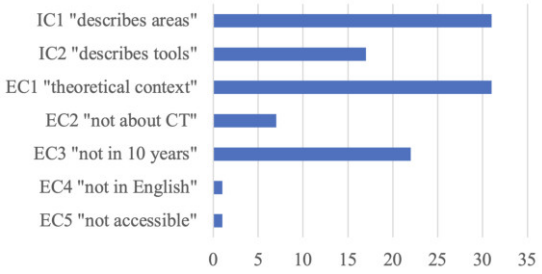

Figure 2: Accepted and rejected primary sources by inclusion and exclusion criteria

3.2.4 Quality Criteria

In addition to the inclusion and exclusion criteria the set of primary studies can be filtered through a quality assessment. For this a quality assessment checklist is formulated and applied on the papers remaining after the previous step where the inclusion and exclusion criteria are applied. In this step the papers are read more thoroughly. The checklist written for this paper contains the following items.

- Are findings sufficiently supported (data, sources) and credible?
- Are problems (limitations, threats to validity) discussed?
- Are findings relevant and provide new value in context of industry?
- Does the paper provide answers to a research question?

These questions aim at accessing the reporting, rigor, credibility and relevance of the findings [10] as well as the direct relevance of the content for the topic. We chose to include questions regarding the relevance in this step of the process again, despite them being already asked in the inclusion/exclusion criteria, because in this step the papers are examined much closer, so a premature decision based on a misleading abstract can be remedied. To simplify the evaluation the questions are answered with *Yes*, *Partially*, *No*, where Parsifal assigns a score *1*, *0.5* and *0* respectively to the answers and sums them up per paper, to get an overall quality score. Papers with an overall quality below *2.0* are excluded from the further process to focus resources on papers of higher quality and remove those that fail in multiple quality categories and specifically eliminate those papers that failed in the last category on relevance to the given topic.

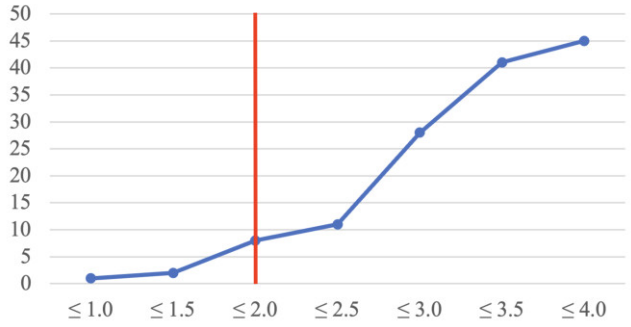

Figure 3: Number of primary sources per quality scores and the cut off line in red

Figure 4: Amount of primary sources remaining after each step of the process to collect sources

3.3 Search Results

After applying the search queries to the search engines 190 primary sources were returned. More than half of those were found in the *ISI Web of Science* and the *IEEE Digital Library* alone as visualized in figure 1. After filtering out a total of 81 duplicates and applying the inclusion and exclusion criteria (see figure 2) 45 papers remained. Those were analyzed regarding their quality, and an additional 8 papers were removed for which the resulting quality score was too low (see figure 3), so that in the end 37 papers remained for further analysis (see figure 4).

3.4 Data Extraction

In order to systematically collect all relevant information from the primary sources and answer the research questions a data extraction form must be designed. This form contains categories that are filled out using each paper itself and they should relate to the research questions [6]. The categories of the data extraction form for this SLR are given in table 2 where the individual categories are listed alongside a description motivating what the category is needed for.

Category	Motivation
Year of publication	This information may help answering *RQ1*. The idea is to see if there is a visible trend in the amount of papers published per year, per domain of industry that CT is applied in.
Authors	This information relates to *RQ1* as well, because if multiple papers are published on certain fields of industry, but they were all published by the same authors, then that would not necessarily indicate that there is a bigger trend.
Summary	A short summary of the paper helps to recall the content of the paper and subsequently supports all research questions.
Domain	The domain of industry relates to *RQ1* and *RQ2*.
Benefits	This relates to the benefits of CT that were mentioned in the given paper and relates to *RQ3*.
Limits	This relates to the highlighted drawbacks and limits of CT that were mentioned in the given paper and relates to *RQ2*.
Requirements	This plays a role in *RQ2* and extends the form item on limits to find out what the challenges of CT are.
Applied tool	If the paper used a specific tool to apply CT then this is of interest for *RQ4*.

Table 2: Data Extraction Form

4. RESULTS

In this chapter the results of the data extraction are summarized in order to answer the research questions.

4.1 RQ1 - In which domains of the industry is CT used and are there any observable trends?

The transportation sector is by far the largest industrial area referenced by the papers. This includes both autonomous driving and railway sector applications. Autonomous driving is referenced by three papers which apply CT to model test scenarios and generate test cases to validate the correct behavior of autonomous driving functionalities [11, 12, 13]. Two of these papers were written by the same group of researchers [11, 12]. The six papers in context of railway systems are focused on validating embedded systems [14, 15, 16, 17, 18, 19]. These papers were written by the same group of authors as well.

Embedded Systems are also referenced in two more papers, however they deal with the programming of hardware on a higher level [20, 21].

Data Management systems make up the second largest sector with six papers [22, 23, 24, 25, 26, 27]. Here some of the research comes directly from companies like SAP [26] and Adobe [23, 24].

Next, we categorised a set of five papers into the broader category of networking [28, 29, 30, 31, 32]. This includes distributed control systems for industry [28, 29], cloud infrastructure [30] and IoT [31]. Finally, three papers refer to Web and Mobile APP development [33, 34, 35].

Overall, the domains are very diverse. CT is widely applicable in environments where comprehensive testing is necessary for safety reasons or where a large input parameter space must be tested. However, there is no larger trend recognizable. The number of individual papers for each category is too low to evaluate the temporal distribution to make a definite observation and while it seems initially like there may be a trend in the application in the transportation domain by comparing the number of papers in this category it turns out that those papers come from the same group of researchers.

4.2 RQ2 - What are the challenges of applying CT in industry?

Preparation and effort: One of the largest problems reported when applying CT in industry is the upfront effort required to integrating it into the process. This can be resource-intensive in terms of time and effort [36], because the effective application requires specialized tools and expertise [30, 36] and may have a steep learning curve [17, 37]. Also, the initial effort needed to create the model from which the test cases are generated from is a problem [19, 37]. This is especially for large and complex systems [27] with complex constraints and interactions [32, 38]. Despite the effort, the problem with the modelling process is also that substantial knowledge of the system is required to accurately model parameters and their relationships [17]. Even in approaches that try to automate the modelling process, knowledge of the system is still highly important, as still individual steps are required or recommended to improve the model [39] for example to add information on invalid parameter combinations [20]. Also, those automated processes still require inputs, like initial test suites to discover parameters and possible values [31, 40]. Not just the model generation takes effort and domain knowledge upfront however, also work is needed to implement abstract test cases [11], and complete the expected outputs [15, 38].

Coverage and Computation: It is important to note that CT does not guarantee good coverage either. First of all, CT is not actually exhaustive, so there are not actually tests for all possible scenarios and some remain untested [28] and not every fault can be found [36]. Secondly, the ability to generate meaningful test cases depends heavily on the quality of the underlying input model [30, 41, 42], for example if fault triggering parameters are not included in the model, they could not be considered by CT [32]. Lastly, the coverage depends on the chosen interaction strength t, but choosing a higher testing strength, causes its own set of issues, like a drastic increase in the number of generated test cases [15]. This will lead to an increase in computational resources to generate the test cases [37, 43] which may strain testing resources [32] and additional test cases may make the approach impractical [18, 19]. In some cases it was reported that a testing strength of four made the application of CT already infeasible [23] or impractical [41]. For pairwise test suites a study found that experienced test engineers can outperform CT in terms of the size of their test suite relating to detection capabilities [16].

In summary, CT requires many resources to make the testing engineers familiar with the process and tools, expert knowledge is needed to generate an input model and implement the resulting test cases. This process can however be supported by automation in some cases. Finally, the quality of the resulting test suite depends on various factors, like the quality of the input model, and the testing strength, which may make the resulting test suite too large to manage.

4.3 RQ3 - What are the benefits of applying CT in industry?

The papers mentioned a variety of reasons why CT is applied in industry. These are described in more detail in the following.

Enhanced Reliability: CT is adept at identifying faults that are caused by interactions among multiple factors. Traditional testing methods might miss these faults because they often test factors in isolation or with limited combination [32, 37, 39, 41, 43, 44]. CT ensures that the software behaves correctly under various conditions by systematically covering combinations of input parameters [38, 45]. This leads to improved reliability of the software as it ensures that interaction faults are detected and addressed during the testing phase [36]. It provides better coverage and significant effectiveness in detecting multi-factor faults, often missed by other testing techniques [33].

Multi-factor Fault Detection: CT is particularly effective in detecting faults that occur only due to specific combinations of inputs, which are often complex and hard to predict [38, 41, 45]. By testing multiple combinations systematically, CT catches these complex bugs early in the development cycle. Especially applying CT early in the development life cycle the detection of such complex bugs prevents them from causing issues in the production environment, saving time and resources that would be needed for later bug fixes [20, 32].

Reduction in Test Cases: CT techniques generate a smaller number of test cases that can still cover a wide range of scenarios[14, 15, 20, 37, 44]. This is achieved by focusing on combinations of parameters rather than testing all possible combinations exhaustively [38]. This significantly reduces the time and resources required for testing without compromising on the thoroughness of the test coverage [26, 29, 46].

Optimized Testing Effort: CT maximizes test coverage with minimal effort by ensuring that each test case covers as many combinations of parameters as possible [13, 46]. This is particularly beneficial in environments where resources are limited or project deadlines are tight. Efficient use of testing resources allows for more thorough testing within the same time frame, improving the overall quality of the software [28, 37, 47].

Comprehensive Coverage: CT ensures that a wide range of input parameter combinations are tested, including edge cases and unlikely scenarios [21, 22, 45]. This comprehensive approach ensures that even rare interactions are considered. This leads to higher confidence in the robustness and reliability of the software, as it is less likely to fail due to untested interactions [21, 36].

Scalability: CT can handle complex systems with numerous input parameters without facing the state space explosion problem on a scale that exhaustive testing would. This scalability is achieved through techniques that focus on the most significant combinations of parameters [19, 23, 28, 37]. It is suitable for systems with high interdependencies among parameters, where other testing methods might struggle [17, 32].

Versatility: CT is versatile and can be applied in various domains, including software security in distributed systems and automated control systems [28, 41]. This versatility makes it a valuable testing strategy for a wide range of applications. Its ability to adapt to different testing scenarios and requirements makes CT a useful tool across industries [14].

4.4 RQ4 - What tools are used to apply CT in industry?

Out of the selected 38 papers, 21 reference a tool used to apply CT. seven Papers [13, 19, 22, 24, 32, 41, 45] referenced ACTS. The tools CCM [18, 22], CAMETRICS [38, 45], CAGEN [15, 45], PICT [20, 30], Avocado [39, 46] occur in two papers respectively. The tools Robusta [27], Allpairs [33], Sysmodis [35], Q-EMCQ [14], Seafox [16], VCM-SL [13] are

referenced in one paper respectively.

Besides, some paper also introduce their own custom approaches to generate combinatorial models based on code [20] and design artifacts [31], legacy tests [40, 47] and to generate combinatorial test cases [21, 23, 37].

Automated Combinatorial Testing for Software (ACTS): ACTS [13, 19, 24, 45] generates test cases based on combinatorial methods. It supports different combinatorial strategies such as pairwise testing, t-way testing, and mixed-strength testing. It helps reduce the number of test cases while ensuring high coverage [32]. It is widely adopted in various industries to streamline the test generation process and improve the efficiency and effectiveness of software testing.

Pairwise Independent Combinatorial Testing (PiCT): Developed by Microsoft, PiCt [20, 30]is a tool for generating combinatorial test cases focusing on pairwise combinations. Known for its efficiency and ease of use, PiCT supports various constraints and parameter types, making it versatile for different testing scenarios. It is used extensively in the industry to optimize testing processes by ensuring critical pairwise interactions are covered with minimal test cases. TEPACS [20] a tool developed for CT that uses PiCT, is a framework for bug detection in pragma-related bugs in high-level synthesis tools used for field-programmable gate array design.

ReACP: A semi-automated framework, ReACP can reverse-engineer the Access Control Policies of web applications with the goal of validating them. This solves the problem where Access Control is often hard-coded in business logic and poorly documented. The tool explores web applications, mines input specifications from access logs, and generates more access requests using CT generation. Machine learning is used to learn policies and roles. The approach was validated on four web applications, where 97.8% of learned policies were correct, and 64 Access Control issues were detected and reported [33].

CAGEN: A combinatorial test generator that automates the creation of test cases for software testing. CAGEN [15, 45] generates test cases that cover all possible combinations of input parameters, ensuring thorough testing coverage.It supports higher-index arrays and user-defined constraints, minimizing redundancy and maximizing efficiency in test generation [45]. Extensively used in various industries, it seamlessly integrates into existing testing frameworks and handles large-scale testing scenarios effectively.

CAMETRICS: A tool designed for advanced combinatorial analysis and measurement of test sets. It focuses on using CT techniques to improve the efficiency and effectiveness of software testing. The tool helps in generating and analyzing test suites, aiming to detect faults more effectively by considering various combinations of input parameters [38]. It is used for combinatorial coverage analysis mainly focusing on large object-oriented applications in the .NET framework [45].

AVOCADO: An open-source framework designed to facilitate constrained interaction testing for industrial applications. AVOCADO integrates tools and libraries for creating, executing, and evaluating automated tests [39, 46].

ROBUSTA: A test selection strategy used to implement Combinatorial Robustness Testing by considering exclusion- and error-constraints to avoid input masking and ensure thorough testing of both normal and exceptional behaviors of the system [27].

SYSMODIS: It uses covering arrays for systematic sampling and employs machine learning algorithms to analyze test results and discover likely guard conditions for transitions between screens [35] .

5. THREATS TO VALIDITY

In this chapter we want to briefly discuss points of weakness of this SLR, to highlight aspects that could be improved in later research.

Firstly, we made the decision to focus on more recent research and eliminated 22 potential primary sources by introducing the exclusion criterion to reject papers that were not published in the last 10 years (see Chapter 3.2.3). This decision was motivated to lay focus on more recent research, however it made it more difficult to make judgements of the trends in the industry (see Chapter 4.1). Another problem worth noting is that the steps in the rounds to eliminate papers, such as the application of the inclusion and exclusion criteria (see Chapter 3.2.3) and the quality assessment (see Chapter 3.2.4) were made by a single author. Similarly, when it comes to the data extraction (see Chapter 3.4) the set of remaining papers was split between both authors and individual assessments were made and not counter validated. To increase confidence into the findings and reduce the bias of individual authors, it would be better to assess each paper individually, compare the findings and discuss diverging assessments.

6. SUMMARY

The SLR demonstrates that CT is not merely a theoretical approach but is practically applied over a diverse range of industries (see Chapter 4.1 RQ1). However, the application is not trivial; test engineers must first become familiar with the tools, and significant domain knowledge and time are necessary to create input models, which heavily influence the quality of the combinatorial test suite. Also, depending on the required testing strength, this approach may become impractical in terms of resources (see Chapter 4.2). Despite these challenges, the benefits of applying CT in the industry are substantial, including a reduction in the number of test cases, improved detection of interaction faults, increased efficiency, and cost-effectiveness (see Chapter 4.3). Additionally, the tools like CAGEN, CAMETRICS, AVOCADO, ROBUSTA, and SYSMODIS facilitate the application of CT by automating test generation, supporting combinatorial coverage analysis, and enabling constrained interaction testing in industrial applications. These tools help in minimizing redundancy and maximizing efficiency in test generation, making CT a valuable technique for improving the efficiency and effectiveness of software testing (see Chapter 4.4).

7. REFERENCES

[1] John C. Knight, Kevin G. Wika, and Shannon Wrege. Exhaustive testing as a verification technique †. 2008.

[2] Athira Varma Jayakumar, Richard Kuhn, Brandon Simons, Aidan G. Collins, Smitha Gautham, Rick Hite, Raghu N. Kacker, Abhi D. Rajagopala, and Carl R. Elks. A pseudo-exhaustive software testing framework for embedded digital devices in nuclear

power. *12th Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies (NPIC&HMIT 2021)*, 2021.

[3] Carlos Pacheco. Directed random testing. 2009.

[4] D. Richard Kuhn, Raghu N. Kacker, and Yu Lei. *Introduction to Combinatorial Testing*. Chapman & Hall/CRC, 1st edition, 2013.

[5] D.R. Kuhn, D.R. Wallace, and A.M. Gallo. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30(6):418–421, 2004.

[6] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. 2, 01 2007.

[7] Rob Briner and David Denyer. *Systematic Review and Evidence Synthesis as a Practice and Scholarship Tool*, pages 112–129. 09 2012.

[8] Parsifal, https://parsif.al.

[9] Angela Carrera-Rivera, William Ochoa, Felix Larrinaga, and Ganix Lasa. How-to conduct a systematic literature review: A quick guide for computer science research. *MethodsX*, 9:101895, 2022.

[10] You Zhou, He Zhang, Xin Huang, Song Yang, Muhammad Ali Babar, and Hao Tang. Quality assessment of systematic reviews in software engineering: a tertiary study. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE '15, New York, NY, USA, 2015. Association for Computing Machinery.

[11] Florian Klueck, Yihao Li, Mihai Nica, Jianbo Tao, and Franz Wotawa. Using ontologies for test suites generation for automated and autonomous driving functions. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 118–123, 2018.

[12] Florian Klück, Yihao Li, Jianbo Tao, and Franz Wotawa. An empirical comparison of combinatorial testing and search-based testing in the context of automated and autonomous driving systems. *Information and Software Technology*, 160:107225, 2023.

[13] Jaganmohan Chandrasekaran, Yu Lei, Raghu Kacker, and D. Richard Kuhn. A combinatorial approach to testing deep neural network-based autonomous driving systems. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 57–66, 2021.

[14] Bestoun Ahmed, Eduard Enoiu, Wasif Afzal, and Kamal Zamli. An evaluation of monte carlo-based hyper-heuristic for interaction testing of industrial embedded software applications, 02 2020.

[15] Muhammad Nouman Zafar, Wasif Afzal, and Eduard Enoiu. Evaluating system-level test generation for industrial software: A comparison between manual, combinatorial and model-based testing. In *2022 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 148–159, 2022.

[16] Peter Charbachi, Linus Eklund, and Eduard Enoiu. Can pairwise testing perform comparably to manually handcrafted testing carried out by industrial engineers? In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 92–99, 2017.

[17] Takashi Kitamura, Akihisa Yamada, Goro Hatayama, Shinya Sakuragi, Eunhye Choi, and Cyrille Artho. Classification tree method with parameter shielding. pages 230–241, 08 2017.

[18] Miraldi Fifo, Eduard Enoiu, and Wasif Afzal. On measuring combinatorial coverage of manually created test cases for industrial software. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 264–267, 2019.

[19] Sara Ericsson and Eduard Enoiu. Combinatorial modeling and test case generation for industrial control software using acts. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 414–425, 2018.

[20] He Jiang, Zun Wang, Zhide Zhou, Xiaochen Li, Shikai Guo, Weifeng Sun, and Tao Zhang. A testing program and pragma combination selection based framework for high-level synthesis tool pragma-related bug detection. *IEEE Transactions on Software Engineering*, 50(4):937–955, 2024.

[21] Bestoun Ahmed, Mouayad Sahib, Luca Maria Gambardella, Wasif Afzal, and Kamal Zamli. Optimum design of pid controller for an automatic voltage regulator system using combinatorial test design. *PLoS ONE*, 11:1–22, 11 2016.

[22] Murat Ozcan. Applications of practical combinatorial testing methods at siemens industry inc., building technologies division. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 208–215, 2017.

[23] Darryl Jarman, Riley Smith, Gabe Gosney, Ludwig Kampel, Manuel Leithner, Dimitris Simos, Raghu Kacker, and Richard Kuhn. Applying combinatorial testing to large-scale data processing at adobe. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 190–193, 2019.

[24] Riley Smith, Darryl Jarman, Jared Bellows, Richard Kuhn, Raghu Kacker, and Dimitris Simos. Measuring combinatorial coverage at adobe. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 194–197, 2019.

[25] Iram Arshad, Saeed Alsamhi, and Wasif Afzal. Big data testing techniques: Taxonomy, challenges and future trends. *Computers, Materials Continua*, 74, 11 2021.

[26] Artur Andrzejak and Thomas Bach. Practical amplification of condition/decision test coverage by combinatorial testing. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 341–347, 2018.

[27] Konrad Fogen and Horst Lichter. An industrial case study on fault detection effectiveness of combinatorial robustness testing. 12 2020.

[28] Peter W. V. Tran-Jørgensen, Tomas Kulik, Jalil Boudjadar, and Peter Gorm Larsen. Security analysis of cloud-connected industrial control systems using combinatorial testing. In *Proceedings of the 17th*

ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE '19, New York, NY, USA, 2019. Association for Computing Machinery.

[29] Cheng Zong, Yanliang Zhang, Yongming Yao, Shiyong Shuang, and Ziyuan Wang. A comparison of fault detection efficiency between adaptive random testing and greedy combinatorial testing for control logics in nuclear industrial distributed control systems. *IEEE Access*, 9:84021–84033, 2021.

[30] Abdul Rauf EM and E.Madhusudhana Reddy. Combinatorial testing: A case study approach for software evaluation. In *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–5, 2015.

[31] Dale Blue, Orna Raz, Rachel Tzoref-Brill, Paul Wojciak, and Marcel Zalmanovici. Proactive and pervasive combinatorial testing. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '18, page 144–152, New York, NY, USA, 2018. Association for Computing Machinery.

[32] Itai Segall, Rachel Tzoref-Brill, and Aviad Zlotnick. Combining minimization and generation for combinatorial testing. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–9, 2015.

[33] Ha Thanh Le, Lwin Khin Shar, Domenico Bianculli, Lionel Claude Briand, and Cu Duy Nguyen. Automated reverse engineering of role-based access control policies of web applications. *Journal of Systems and Software*, 184:111109, 2022.

[34] Brian Elgaard Bennett. A practical method for api testing in the context of continuous delivery and behavior driven development. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 44–47, 2021.

[35] Omer Korkmaz and Cemal Yilmaz. Sysmodis: A systematic model discovery approach. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 67–76, 2021.

[36] Xuelin Li, Ruizhi Gao, W. Eric Wong, Chunhui Yang, and Dong Li. Applying combinatorial testing in industrial settings. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 53–60, 2016.

[37] Muhammad Zafar, Wasif Afzal, and Eduard Enoiu. An empirical evaluation of system-level test effectiveness for safety-critical software. 04 2023.

[38] Manuel Leithner, Kristoffer Kleine, and Dimitris E. Simos. Cametrics: A tool for advanced combinatorial analysis and measurement of test sets. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 318–327, 2018.

[39] Bestoun S. Ahmed, Amador Pahim, Cleber R. Rosa Junior, D. Richard Kuhn, and Miroslav Bures. Towards an automated unified framework to run applications for combinatorial interaction testing. In *Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering*, EASE '19, page 252–258, New York, NY, USA, 2019. Association for Computing Machinery.

[40] Rachel Tzoref-Brill and Shahar Maoz. Modify, enhance, select: co-evolution of combinatorial models and test plans. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 235–245, New York, NY, USA, 2018. Association for Computing Machinery.

[41] Linghuan Hu, W. Wong, D. Kuhn, and Raghu Kacker. How does combinatorial testing perform in the real world: an empirical study. *Empirical Software Engineering*, 25, 07 2020.

[42] Marcos Palacios, José García-Fanjul, Javier Tuya, and George Spanoudakis. Automatic test case generation for ws-agreements using combinatorial testing. *Comput. Stand. Interfaces*, 38(C):84–100, feb 2015.

[43] Dong Li, Linghuan Hu, Ruizhi Gao, W. Eric Wong, D. Richard Kuhn, and Raghu N. Kacker. Improving mc/dc and fault detection strength using combinatorial testing. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 297–303, 2017.

[44] Bestoun S. Ahmed. Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing. *Engineering Science and Technology, an International Journal*, 19(2):737–753, 2016.

[45] Christian Eitner and Franz Wotawa. Crucial tool features for successful combinatorial input parameter testing in an industrial application. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 188–189, 2019.

[46] Jan Richter, Bestoun S. Ahmed, Miroslav Bures, and Cleber R. Rosa Junior. Avocado: Open-source flexible constrained interaction testing for practical application. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 185–190, 2020.

[47] Marcel Zalmanovici, Orna Raz, and Rachel Tzoref-Brill. Cluster-based test suite functional analysis. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 962–967, New York, NY, USA, 2016. Association for Computing Machinery.

Modernization: Literature Analysis of the Risk Related to Modernization Projects

Ilinca Baicu
RWTH Aachen University
ilinca.baicu@rwth-aachen.de

Shu Zhang
RWTH Aachen University
shu.zhang@rwth-aachen.de

ABSTRACT

Software modernization aims to replace outdated legacy systems with more advanced and efficient technologies that will better support the future business requirements of organizations. The process of modernization involves updating legacy software with newer languages, frameworks, and infrastructure platforms. As market demands, customer expectations, and industry trends change, organizations require continuous business expansion and product innovation to maintain their position in the marketplace. Modernization projects are crucial for promoting technical progress, boosting operational efficiency, and adapting to changing market trends, thereby supporting organizational development and sustainability across various industries. Modernization offers numerous advantages, including enhanced performance, security, and agility. However, it also introduces risks that can challenge the success of projects and organizational resilience. While risks associated with modernization have been acknowledged, the existing literature on this subject is fragmented. It tends to concentrate on specific aspects without providing a comprehensive global overview, making it difficult for organizations to understand and apply modernization processes. To address the gap, this paper examines the risks associated with modernization by analyzing existing literature on the subject. Through a review of various case studies, academic research, and industry reports, it aims to identify and categorize the risks of software modernization into distinct groups and sub-groups. This structured approach is designed to help organizations comprehend the various types of risks involved and their potential impact on projects. In the future, organizations can use this risk categorization to prioritize resources more efficiently based on the significance and potential impact of each risk.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Software Engineering**]: Management—*productivity, programming teams, software configuration management*

Keywords

Modernization, Risk, Risk Categorization, Business Risk, Technical Risk, Human-affiliated Risk, Project Life Cycle Risk, Impact-based Risk

SWC Seminar 2024 RWTH Aachen University, Germany.

1. INTRODUCTION

1.1 Background

A legacy system is a computer system that, despite being slated for replacement by a more advanced system, remains in use [20]. The advantage of such systems lies in their established functionality and interfaces. However, the downside is that these systems retain their vulnerabilities and shortcomings, potentially negatively impacting the daily operations of organizations [4]. As the pressure to modernize systems and processes increases in today's technology-driven business environment, the necessity for organizations to modernize their software systems becomes even more important [5]. Software modernization involves updating or replacing legacy systems with newer technologies. Modernization is often called by a different name, including *replacement, migration, renovation, recasting* and *revamping*. They refer to a range of activities to transition from legacy systems to modern infrastructures [3], making the system adaptable to current technical standards and business requirements. Software modernization aims to enhance various aspects of a system, including (but not limited to) maintainability, scalability, and the integration of new technologies. It can boost product performance, strengthen security and provide an agile system for researchers. For organizations, modernization could lead to better productivity, lower costs, and improved customer satisfaction [22]. Moreover, modernization enables organizations to retain the knowledge and functionality included in legacy systems, which are important organizational investments [1].

However, as with any software project, modernization projects also involve risks that can undermine its success. Risks of modernization refer to the potential problems, challenges, and adverse outcomes that can arise when organizations update or replace their legacy systems and processes with newer technologies or methods. They should be considered as early as possible and managed appropriately. A reason for taking risks seriously is that around 50% of modernization projects fail [1]. Modernization projects can bring

a lot of benefits, but organizations do not always fully manage the risks associated with them, leading to delays, budget overruns and even project failures.

In this paper, the risks that may arise during the modernization process are explored in detail by analyzing the currently existing literature on the topic. By investigating various case studies, academic research, and industry reports, this paper seeks to provide a comprehensive understanding of the risks and challenges associated with software modernization. Specific risks that are commonly identified in modernization projects are being categorized to help organizations manage risks in a more systematic and framed approach, which allows them to be more comprehensive in their response to the modernization of legacy systems.

The complexity and challenges of software modernization require organizations to be familiar with the risks they face in the process. As software modernization becomes increasingly essential to maintaining a competitive advantage, ensuring operational efficiency and increasing the ability to utilize new technologies, understanding and mitigating the related risks is the key to guaranteeing the success of a modernization project.

This paper aims to answer the following two research questions:

- **RQ1: What risks occur in the process of software modernization?**
- **RQ2: How can these risks be categorized?**

2. RELATED WORK

In a competitive market environment, fast-responding software systems are a vital characteristic for organizations to improve their operational efficiency and reduce technical debt [22]. However, modernization projects face a variety of risks that can significantly influence their success. It is critical to understand these risks and develop effective mitigation strategies to ensure success. This section aims to provide a comprehensive overview of the general concepts of software modernization, specific technologies, and understanding of modernization risks to establish a foundation for our follow-up analysis and discussion.

2.1 Context of modernization

In today's world, science and technology are continuously developing, which also requires software to be constantly updated and functions to be added to help humans make it more adaptable to the modern environment [5]. Legacy systems sometimes have high maintenance costs and lack modern functionality due to outdated software versions, complex architecture, inefficient code, and hardware limitations. Therefore, they are not easily adaptable to the constantly growing customer demands and developer workloads [14]. Modern software can increase efficiency, lower operational expenses, and improve performance for organizations requiring advanced digital technologies to drive business transformation. For enterprises, modernized software can provide the support of a digital infrastructure that facilitates the automation of business processes to enhance the user experience and allow for data-driven decision-making [12].

In general, software modernization provides a strategic approach for organizations to revitalize legacy systems and develop them into more broadly supported and efficient platforms or systems. It allows organizations to leverage existing investments in order to remain competitive and innovative in the current digital environment [19]. Modernization is considered one of the components describing the transition of the system from an old one to a new one [3]. Barbier and Recoussine [3] introduced five other applicable objects: replacement, migration, renovation, recasting and revamping.

The increase in rapid changes in the technology environment promotes the update of organizations' software before the system becomes obsolete. Hence, organizations are taking on more software modernization projects [5]. When an existing application is too outdated or inefficient, organizations may consider rebuilding the entire application with the latest technology from scratch [6]. Although rebuilding offers excellent potential for innovation and the possibility of significant improvements in the functionality and performance of software, it is also associated with being resource-intensive, having high costs, being time-consuming, and representing a high risk in general. Cloud migration is another common approach used by organizations to achieve modernization. The wide range of cloud services and solutions offered by leading cloud computing providers makes it an important platform for digitization and modernization of organizations. Cloud migration primarily includes assessment and planning, data migration, deployment and transformation, testing and verification, optimization, *etc.* Similar to the cloud, microservices architecture and DevOps (development and operations) complement each other. DevOps makes it possible to develop, test, and deploy these services rapidly [2]. With the decoupled components and automated delivery pipeline introduced by DevOps, teams can update functions more regularly and respond faster to evolving feedback from users. Indeed, DevOps encourages various teams to collaborate and share responsibilities throughout the software life cycle by providing different technical tools and processes [17].

2.2 Risks of modernization

Modernization projects involve updating and adapting legacy systems, which are important for improving the efficiency and competitiveness of organizations [19]. Undergoing a modernization process does not come without risks. Finding and solving risks play a decisive role in the success rate of the modernization project, especially as these projects involve updating and adapting legacy systems. For code in legacy systems, developers could misunderstand the purpose of the legacy code, which brings the risk of introducing disruptive functionality to organizations [18]. A comprehensive assessment of complex and under-documented legacy systems is also time-consuming. One significant risk is the incompatibility between legacy systems and modern technologies. Givehchi *et al.* [9] described that the integration of new technologies, such as big data analytics and service-oriented architectures, with legacy systems, might be challenging due to the heterogeneity of industrial systems. In the modernization process, more information and data is being transferred to cloud platforms. Stavru *et al.* [21] discussed the challenges of software modernization and cloud computing from an infrastructure perspective. Hussein [10] pointed out that data migration security and privacy issues are being questioned. Risks affiliated with humans are also one of the aspects that organizations should focus on.

Khadka *et al.* [11] emphasized that the lack of modern technological expertise among existing employees could hinder the success of modernization projects. Rashid *et al.* [18] pointed out that employees who are used to an old system resisting the change are another major risk. Practical modernization training is essential for organizations to address these risks. According to Fleurey *et al.* [7], the financial burden of modernization can be significant. In particular, testing costs may account for the majority of the overall system costs. In the beginning, organizations with limited financial resources may have difficulty allocating enough funds for modernization [24]. During the transition period, disruptions in the organization's business could have a negative impact on customer satisfaction. Teppe *et al.* [26] emphasized the importance of reducing costs and the role and resources that testing plays in the migration process by describing a user case of an IT solution in the travel industry. Most literature related to the risks of software modernization considers the challenges organizations face. In order to address the risks, organizations require a multifaceted approach that includes thoughtful planning, stakeholder activation, *etc.* Generalizing and categorizing based on current research can be more helpful for modernization.

3. METHODOLOGY

This paper adopts a predominantly literature-based research approach, utilizing existing publications as a foundation for both the derivation of this paper's results and the formulation of conclusions. Central to this methodology is the construction of risk categories, which are derived from a comprehensive analysis of prior studies and reports. The risks are critically assessed and categorized based on common characteristics and the nature of the risks they encompass. This approach aims to enhance the understanding of how different risks interact within broader risk landscapes. Each category will be defined and justified based on the literature, with criteria for inclusion and exclusion, providing a basis for future researchers and practitioners to follow.

The primary data for this study consists of peer-reviewed journal articles, conference papers, books, and online resources from big technology companies. Databases such as *ResearchGate, IEEE Xplore* and *Google Scholar* were extensively used. A structured search strategy was employed to gather literature. Keywords and phrases related to risks in software modernization, such as *software modernization, risks in software modernization, risk categories in modernization, technical risks, software business risks, technical modernization risks, legacy system risks, teamwork risks in modernization, data loss in modernization, software organization modernization, security risks, software project risks* were used. The boolean operator (AND) was also utilized to refine and broaden the search. For instance, the search "security AND data loss risks" was used to ensure that the gathered documents specifically addressed both concepts simultaneously. In addition to our primary sources, the references cited within these sources were also regarded. This method allowed the expansion of the literature base by incorporating relevant studies and papers referenced in the initial set of sources. Due to numerous publications regarding software modernization and its risks, inclusion and exclusion criteria were clearly defined to help with the selection of relevant literature. The inclusion criteria contained studies published in English, peer-reviewed articles — due to their rigorous review process — and studies incorporating the keywords mentioned above. Only studies that directly address the research questions or closely related topics were included. Furthermore, to ensure the inclusion of the most current information, we focused on studies published within the last ten years. However, older studies were included if they provided a foundational understanding or historical perspective relevant to the research questions. It is also a fact that both quantitative and qualitative studies were taken into account. In hopes of ensuring the quality of the chosen literature, high-quality evidence, such as systematic reviews, meta-analyses, and well-conducted empirical studies, were prioritized. Exclusion criteria involved brief abstracts or summaries without full texts, non-peer-reviewed articles, publications not taking modernization risks into account or publications where none of the keywords were present. To ensure the relevance of the selected publications, the abstract served as the initial point of guidance. This was supplemented by an examination of the introduction and a cursory review of the results section. Subsequently, a deeper dive into each chapter and the paper as a whole was conducted. The focus while assessing the literature was set on the abstract, introduction, result and discussion sections. These chapters were specifically chosen because they offer a comprehensive overview of the paper's content and represent its core elements. Data extraction was conducted using a standardized form and software such as *Zotero and Citavi.* This form for how data is kept was used to systematically record key information from each source, such as author(s), year of publication, research objectives, methodologies, findings, and conclusions. This facilitated an organized synthesis of the literature, ensuring that relevant data was considered and appropriately analyzed. This chapter outlined the methodologies used to undertake a systematic review of the literature on risks in software modernization. The subsequent chapters will utilize this foundation to explore possible categories and groups of risks when undergoing a modernization process.

4. RESULTS

This section discusses common risks encountered in software modernization projects based on literature. In order to create categories and group the identified risks, a method of assessment is required. To identify the various categories, focal points, and bottlenecks in the modernization process were analyzed. After defining these categories, each identified risk was evaluated and assigned to a specific category. In this paper, different types of categorization and several associated risks are considered. These are presented in this chapter as well.

4.1 Technical, business, and human-affiliated risks

Technical, business, and human-affiliated risks in software modernization are the primary categories of risks that can impact the success of such projects. Technical risks involve challenges related to technology, business risks affect the business aspects, including potential disruptions to operations and misalignment with business goals, while human-affiliated risks pertain to the people involved in the project. This category aims to provide a comprehensive framework and emphasizes the importance of integrating technical advancements, aligning business objectives, and managing hu-

man factors to mitigate risks and leverage opportunities effectively.

4.1.1 Technical risks

This category has been built based on modernization risks that can and should be solved by developers or IT-affiliated workers within a company. Yamashita and Aiko [28] recognize that in the context of software modernization, the emphasis predominantly lies on technical challenges and the proposal of methodologies and tools.

- **Lack of documentation**: When legacy systems lack comprehensive documentation of their architecture, functionality and logic, it is difficult for development teams to understand how these systems fully work. Unannotated legacy code makes it challenging for developers to understand the logic and intended functionality of each code segment. This brings the risk of introducing destructive functionality or making incorrect assumptions to the organization [18].

  Modernization projects encounter ambiguous requirements, especially when stakeholders have different opinions. Without a clear and concise requirements document, there is a risk of miscommunication between stakeholders and teams, which are discussed under the category 4.1.3 [4].

- **Data migration**: Migrating large quantities of data from legacy systems to modern platforms effectively while ensuring data completeness and accuracy can be a challenge for organizations without losing much information [8]. Data migration can exacerbate data quality problems in legacy systems, as inaccurate data could be propagated to the new system, compromising the reliability of modernized software. Furthermore, data security and privacy issues are a primary concern while migrating data to a cloud environment during modernization [10]. Data migration must meet regulatory requirements, local data privacy laws, and industry standards for transmission. Organizations must ensure data security throughout the migration process and prevent data leaks.

- **Security vulnerability**: Legacy systems generally use outdated technologies and frameworks that may have known security vulnerabilities. Although modernization projects update the technology, legacy code may still contain vulnerabilities that can be exploited by attackers. Furthermore, Arpinen and Oskari [1] discuss security-related issues using the example of undergoing modernization with sensitive data being stored in Excel workbooks during the process. Modernized systems may be susceptible to new security vulnerabilities or attack vectors [13]. Security vulnerability can result in substantial profit losses and damage to customer credibility. Additionally, addressing such vulnerabilities often requires a large budget, which can cause further issues for the organization. Therefore, security vulnerabilities are not just a considerable technical risk but also a business risk.

- **Unknown dependencies**: Unknown dependencies in software modernization refer to hidden or unforeseen links between different parts of a system, which may not be apparent until the modernization process is underway. These dependencies can involve various elements such as data, code, infrastructure, or even external systems. When they are not initially identified by the developer team, they can lead to significant challenges later on [28].

4.1.2 Business risks

The importance of considering this category is given by Khadka *et al.* [11], as their work states that legacy systems are business critical. Legacy systems are viewed as the „core systems of the organizations and their failure can result in serious consequences for daily business“ [11]. Seacord *et al.*[20] recognize the importance of business objectives while developing software and, therefore, also while going through the process of modernization. They state in their work that „[a]ny modernization effort that falls outside of business objectives, as management understands them, is headed for cancellation.“ [20]

This category is based on potential direct risks for the company deciding to undergo software modernization. These risks can have a direct negative impact on several aspects of the firm, such as organizational objectives and financial performance. Khadka *et al.* highlight the importance of cost and flexibility within the company by stating: „The primary aim of software modernization is to reduce maintenance cost and increase flexibility“ [11].

- **Cost of testing**: The modernization of complex software systems typically requires extensive testing to ensure functionality and compatibility after the changes are made. Testing for compatibility with different platforms, browsers, devices, and operating systems is critical to minimizing compatibility issues. If organizations do not consider this in advance in their planning and budgeting, the cost of testing could increase. Modernization typically involves significant changes to existing systems, such as updating technology stacks or enhancing legacy code, as mentioned above, which have a high complexity and dependencies. Fleurey *et al.* [7] pointed out that testing costs can account for up to 45% of the total migration cost, which indicates that it is a challenge for organizations to ensure comprehensive test coverage. Although testing might be costly, conducting thorough security testing can mitigate security risks for organizations and prevent potential issues in the future.

- **Limitations imposed by preliminary costs and tasks**: Software modernization requires a large initial investment in terms of resources, time and budget. Organizations need to evaluate the investment thoroughly. Inaccurate cost estimation leads to future funding restrictions. Organizations with limited financial resources may also have trouble allocating adequate funds for initial activities, causing delays in the modernization process [16]. Moreover, software architecture, programming languages, and cloud computing necessary for modernization projects can be new fields to traditional, non-IT-affiliated organizations. They need to recruit and train technical professionals actively with relevant expertise. Non-planned hiring makes the organization incur additional costs and exacerbates resource restrictions.

- **Time-consuming tasks**: As discussed by Yamashita and Aiko [28], these risks center around inefficient tasks that frequently entail the need to deal with obscure technical specifics or navigate unfamiliar technologies. Before software modernization, organizations need to conduct a comprehensive evaluation of the existing legacy systems. For example, analyzing the current state of the software, tracing failures, documenting functionality, *etc.*, is time-consuming, especially for complex and under-documented legacy systems [27]. Furthermore, transitioning from legacy systems to modern frameworks could introduce complications. The situation can get worse because of inflexible task scheduling. This forces developers and testers to keep relearning the details of their tasks, which slows the process down.

4.1.3 Human-affiliated risks

This category encompasses factors related to human behavior, interactions, and capabilities that can impact the projects. In every modernization project, several different individuals are involved, working on different aspects but also being required to function as a team following the same goal: the modernization of the legacy system [1]. Therefore, this category is of great importance.

- **Miscommunication and misinformation**: This issue often arises from workers' ignorance or a lack of knowledge, which can lead to an increase in technical debt. For instance, the introduction of erroneous data into a project, or other human-affiliated errors, can cause delays and adversely affect the entire software modernization effort [28].

- **Lack of knowledge**: Both the lack of solutions for problems that occur and insufficient information pose various risks during the modernization process. Khadka *et al.* [11] illustrate a prime example of information scarcity, where key individuals involved in modernization may leave the organization or retire before the completion of the process, thereby taking critical knowledge with them.

- **Modernization not being accepted by workers/clients/ stakeholders**: Modernization changes workflows and roles within an organization. Employees will resist these changes if they believe that they will disrupt their established ways of working [4], which can lead to decreased productivity and user satisfaction. Another challenge is how to train the workforce with the knowledge of modernization properly [23]. When organizations undergo modernization without training and support, employees will feel overwhelmed by the new technology and lack confidence in their abilities. Stakeholders may consider software modernization as a risky activity with uncertain outcomes [23]. Concerns about project delays, cost overruns, or technical challenges may lead to skepticism or resistance to investing resources for modernization.

- **Project delay**: Mupitra Byan Haqi [15] also discusses the modernization risk associated with the postponement of project milestones beyond their planned completion dates during the modernization process. This can happen due to several human-affiliated reasons, such as faulty planning, conflicts within the team or with the stakeholders, or slow work pace, and it can have a negative impact on several aspects, including customer trust [15].

4.2 Project life cycle risks

Project life cycle risks sort the various types of risks based on the stages they emerge in during the modernization project, from planning to closure. Categorizing risks according to the project life cycle helps in understanding and managing potential challenges specific to each phase of development. In this category, there are three stages presented.

4.2.1 Planning risks

This category refers to risks in defining the scope and objectives of the modernization process, such as stakeholder misalignment or inadequate resource allocation. Effective planning is essential for the smooth execution of a project, as it lays the foundation for all subsequent activities.

- **Inadequate resource allocation:** Resource allocation is crucial in modernization projects as it ensures that financial, human, and technological resources are utilized efficiently and effectively, helping to meet project goals on time and within budget. If not adequately planned before the start of the modernization process, it can lead to complications along the way [25].

- **Stakeholder's perception:** Stakeholder perception greatly influences the success of a modernization process. Positive perceptions can secure necessary support and resources, reduce resistance to change, and enhance project continuity. Conversely, negative perceptions may lead to skepticism, insufficient resource allocation, and challenges in implementing changes effectively [20].

4.2.2 Execution risks

Risks that occur during the actual implementation of modernization, such as technical failures or delays, can negatively impact the entire modernization process. Security threats, data loss, project delays, and miscommunication among project workers, which have been discussed previously in this paper (see chapter 4.1), cannot always be fully mitigated through planning and may arise during the execution of the modernization process.

4.2.3 Closure risks

Closure risks in modernization projects refer to potential issues that could impede the successful finalization and wrap-up of the project. These risks can affect the overall outcomes, leaving some project goals unmet or leading to incomplete integration or deployment. This category may include risks such as lack of documentation, which was already discussed in chapter 4.1 as well as the following:

- **Insufficient testing:** Testing can be complex in modernization projects due to various reasons [4]. In the rush to meet deadlines or to reduce the costs of the modernization project, testing phases can sometimes be shortened. Many steps in performance testing require much manual intervention, so it can be difficult

to complete within a tight time. Shortening of tests might result in software that has not been thoroughly vetted for errors or usability issues, leading to problems when these systems are put into function at the end of the modernization project.

- **Stakeholder dissatisfaction:** If the project outcomes do not align with stakeholder expectations or needs, it could result in a lack of approval or support for the project closure. This misalignment might stem from inadequate communication, changes in stakeholder needs, or failure to meet specified requirements [20].

4.3 Impact-based risks

Impact-based categorization is a method of classifying modernization risks based on the severity of their consequences. This categorization aims to help prioritize responses and allocate resources efficiently by focusing attention and efforts on those areas that can have the greatest impact. Typically, this approach involves dividing risks into high, medium, and low-impact levels. It is important to consider and attempt to prevent or eliminate all risks, regardless of whether they are labeled as high or low level. Each company facing modernization risks is different, and the classification of risks as high or low can be somewhat subjective, depending on the organization's capabilities and focus areas.

4.3.1 High risk

These are risks or elements that, if they materialize, could lead to significant negative outcomes, such as substantial financial loss, severe operational disruptions, or major damage to reputation, potentially threatening the viability of the project or the organization. The following can be considered high modernization risks:

- **Data loss:** Losing critical business data during migration, for example, could have catastrophic effects, including loss of customer trust and legal consequences.
- **Major security breaches:** Introducing new technologies can open vulnerabilities. A major breach could lead to severe legal, financial, and trust damage.
- **Complete project failure:** Due to poor planning or execution, the modernization project fails entirely, wasting invested resources and leaving the organization with an unusable system.

4.3.2 Medium risk

This category refers to risks that can cause moderate disruption but are manageable through effective strategies. As stated before, what is considered a medium risk can depend on the organization undergoing the modernization process.

- **Budget overruns:** Exceeding the budget due to unforeseen complexities or misestimation leads to a need for additional funding, for example, through the cost of testing.
- **Delays in project timelines:** Encountering unexpected issues that push back project milestones, affecting subsequent phases and potentially delaying the overall project completion. This can be caused by the previously mentioned time-consuming tasks, miscommunication among the team members, or even the lack of knowledge and documentation, as the team requires more time to figure out specific steps in the project.

4.3.3 Low risk

This category draws attention to risks with minimal consequences that are unlikely to need significant adjustments to the project but should still be monitored.

- **Minor bugs and glitches:** Small software bugs in the new system that do not significantly impact the functionality or user experience but need eventual fixing and, therefore, require the time and attention of the IT team.
- **Miscommunication and misinformation:** Modernization teams may encounter issues such as misinformation and miscommunication, which can delay the project. However, these problems can be resolved through effective task delegation and clear communication.

By categorizing the risks as presented in this chapter, the research distinctly addresses the multifaceted challenges of software modernization, ensuring comprehensive risk management that covers technical precision, business alignment, and human dynamics, as well as risk levels throughout the modernization process, and the timing of risk emergence in the project.

5. DISCUSSION

As discussed in the preceding chapter, risk categorization can be approached from various perspectives, depending on the focus point of interest. The literature identifies multiple risk types; some may diminish over time, while others may emerge. To address the first research question: *What risks occur in the process of software modernization?*, this paper identifies over 15 specific risks. The presented risks are diverse in origin and require distinct approach strategies. Given the complexity of modernization, it is challenging to encompass all potential risks, particularly as new ones may emerge. Therefore, this paper does not cover all existing risks but focuses on the most frequently cited risks in the literature. The risks are also closely related and influence each other; for example, the high cost of testing potentially results in insufficient testing, which could jeopardize the modernization process.

After carefully considering various categorization methods based on organizations' needs, focus points, and sequence of modernization processes, the response to the second research question in this paper: *How can these risks be categorized?* suggests that risks can be systematically grouped as *technical, business, and human-affiliated risks*, as *part of the project life cycle*, or based on their *impact* on the modernization project. These categories can be further divided in subcategories based on the project life cycle (planning, execution and closure) or by impact (high, medium and low risk).

However, these categories are adaptable and can be customized or expanded based on organizational requirements and subjective assessments. Therefore, it is crucial to recognize that risk categorization can vary significantly based on an organization's specific needs and focus areas and involve a degree of subjectivity. Each category serves a different purpose and is applicable in various scenarios based on the

organization's needs during the modernization project. Although the chosen categorization method is subjective and tailored based on the organization's needs, suggestions as to when to use which category can be made, with the mention that there might be more ways to build categories than the ones presented in this paper.

1. **Technical, business, and human-affiliated risks:** This categorization method is to be chosen when a comprehensive analysis of the software modernization project's impact on the entire organization is needed. This approach is vital when risks in one domain (such as technical) can significantly influence other areas (like business operations or employee productivity). This categorization method is appropriate when integrated planning is required, involving coordination across multiple departments and stakeholders.

2. **Project life cycle risks:** It is to be opted for this categorization method to address risks associated with different phases of the project life cycle, ensuring smooth progression from initiation to closure. This categorization method can be particularly beneficial for organizations that experience challenges at specific stages of a project. For instance, if an organization frequently fails to progress beyond the planning phase due to issues like lack of motivation or inadequate time and resource management, this categorization can help advance the modernization project and ensure its successful completion.

3. **Impact-based risks:** This categorization method can be chosen when the primary focus is on the potential impacts of risks, regardless of their origin. This includes evaluating how different risks might affect key project outcomes such as time, cost, quality, and stakeholder satisfaction. It is beneficial for prioritizing risk management efforts based on the severity of impacts. Especially when resources for the modernization project are limited, this categorization method ensures that critical risks are addressed first. By prioritizing high-level risks, the project can move forward efficiently, while lower-ranked risks can be handled based on the remaining allocated resources. Although low-level risks can be given lower priority and addressed later in the project, they must eventually be determined to ensure a smooth and successful outcome of the modernization process.

It is important to recognize that a clear demarcation of risk categories is challenging due to the interdependent nature of various risks. Different risks often influence and exacerbate one another, creating a complex web of potential issues. To elucidate this phenomenon, it is helpful to revisit the previously mentioned risks and their categories. For instance, security vulnerabilities are classified as technical risks in this paper. However, they are closely intertwined with business risks, such as insufficient testing, potentially due to budget constraints. Moreover, many human-affiliated risks, such as lack of knowledge or miscommunication and misinformation, can result in technical and business risks, such as lack of documentation and increased time-consuming tasks. When considering impact-based risk categorization, it becomes evident that seemingly minor risks, like team miscommunication, can escalate into significant risks, such as data loss or even total project failure. This illustrates the cascading effect where minor issues can lead to substantial consequences. Additionally, in the context of project life cycle categorization, modernization projects often progress through linear stages. Risks encountered during the planning phase can propagate and cause issues in the closure phase, highlighting the importance of addressing risks early and throughout the project life cycle.

6. CONCLUSION & FUTURE WORK

Arpinen's statement "[a]lthough legacy system modernization has been conducted ever since computer systems existed, and it is estimated that over 50 % of system modernization projects fail" [1] underlines the importance of this paper. The high failure rate indicates that organizations often struggle to manage the process or become bogged down in various stages of it, such as planning. By being aware of potential risks and focusing on a categorization method tailored to the organization's needs, one can work on mitigating the identified risks, potentially improving the success rate of modernization.

This paper explored three primary risk categorization methods and classified various potential risks into distinct categories, offering a foundational understanding of the types of risks. As previously discussed, these categories can be interrelated, influencing one another and making clear delineation challenging. The first method categorizes risks into business, human-related, and technical risks. The second method organizes risks based on project phases: planning, execution, and closure. The third method focuses on the impact that risks can have on the project.

However, this paper does not offer extensively discussed solutions or avoidance strategies. While it lays the groundwork by categorizing risks, there remains significant potential for expanding this research to develop robust, dynamic, and context-sensitive modernization risk management strategies. One significant limitation is the lack of development and discussion of standardized responses for different risk categories. Notably, there is a gap in the literature regarding standardized responses for different risk categories, which can evolve with technological advancements. Additionally, while categorizing risks is a helpful first step, there is a limitation in addressing the real-world applicability and effectiveness of these categories. Comprehensive case studies on both successful and unsuccessful software modernization projects are needed to validate the proposed risk categories and to derive practical insights.

These future directions can significantly contribute to the field of software modernization, ensuring more resilient and successful modernization efforts. Therefore, this could be a promising area for future research: risk avoidance and mitigation, presenting a fertile ground for further exploration in risk management. There is an opportunity to develop standardized responses for different risk categories. Future studies could examine how new tools and methodologies, including artificial intelligence and machine learning, impact the mitigation of modernization risks. Further integrating insights from disciplines such as business management, human factors, and organizational behavior would deepen the analysis of non-technical factors that affect modernization risks.

7. REFERENCES

[1] O. Arpinen. Process framework for software modernization decision-making. 2023.

[2] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices architecture enables DevOps: Migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52, 2016. doi: 10.1109/MS.2016.64.

[3] F. Barbier and J.-L. Recoussine. *COBOL Software Modernization: From Principles to Implementation with the BLU AGE® Method*. 01 2015. doi: 10.1002/9781119073147.

[4] B. Batlajery, R. Khadka, A. Saeidi, S. Jansen, J. Hage, et al. Industrial perception of legacy software system and their modernization. *Technical Report Series*, (UU-CS-2014-004), 2014.

[5] H. Bruneliere, J. Cabot, J. L. Canovas Izquierdo, L. O.-E. Arrieta, O. Strauss, and M. Wimmer. Software modernization revisited: Challenges and prospects. *Computer*, 48(8):76–80, 2015. doi: 10.1109/MC.2015.234.

[6] Comella-Dorda, Wallnau, Seacord, and Robert. A survey of black-box modernization approaches for information systems. In *Proceedings 2000 International Conference on Software Maintenance*, pages 173–183, 2000. doi: 10.1109/ICSM.2000.883039.

[7] F. Fleurey, E. Breton, B. Baudry, A. Nicolas, and J.-M. Jézéquel. Model-driven engineering for software migration in a large industrial context. In *Model Driven Engineering Languages and Systems: 10th International Conference, Nashville, USA, 2007. Proceedings 10*, pages 482–497. Springer, 2007.

[8] C. Frisz, G. Brown, and S. Waggoner. Assessing migration risk for scientific data formats. *Int. J. Digit. Curation*, 7:27–38, 2012.

[9] O. Givehchi, K. Landsdorf, P. Simoens, and A. W. Colombo. Interoperability for industrial cyber-physical systems: An approach for legacy systems. *IEEE Transactions on Industrial Informatics*, 13(6):3370–3378, 2017. doi: 10.1109/TII.2017.2740434.

[10] A. A. Hussein. Data migration need, strategy, challenges, methodology, categories, risks, uses with cloud computing, and improvements in its using with cloud using suggested proposed model (dmig 1). *Journal of Information Security*, 2021.

[11] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen, and J. Hage. How do professionals perceive legacy systems and software modernization? In *Proceedings of the 36th International Conference on Software Engineering*, pages 36–47, 2014.

[12] R. Khadka, P. Shrestha, B. Klein, A. Saeidi, J. Hage, S. Jansen, E. van Dis, and M. Bruntink. Does software modernization deliver what it aimed for? a post modernization analysis of five software modernization case studies. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 477–486. IEEE, 2015.

[13] A. Maté, J. Peral, J. Trujillo, C. Blanco, D. García-Saiz, and E. Fernández-Medina. Improving security in nosql document databases through model-driven modernization. *Knowledge and Information Systems*, 63:2209 – 2230, 2021.

[14] A. J. McAllister. The case for teaching legacy systems modernization. In *2011 Eighth International Conference on Information Technology: New Generations*, pages 251–256, 2011. doi: 10.1109/ITNG.2011.51.

[15] B. H. Mupitra. Risk management of process moving from traditional to agile software development. *ACMIT Proceedings*, 4(1):46–51, 2017.

[16] S. Nilsson. Application modernization: Approaches, problems and evaluation, 2015.

[17] D. Pianini and A. Neri. Breaking down monoliths with microservices and devops: an industrial experience report. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 505–514, 2021. doi: 10.1109/ICSME52107.2021.00051.

[18] N. Rashid, M. Salam, R. K. S. Sani, and F. Alam. Analysis of risks in re-engineering software systems. *International Journal of Computer Applications*, 73:5–8, 2013.

[19] B. M. Santos, I. G.-R. de Guzmán, V. V. de Camargo, M. Piattini, and C. Ebert. Software refactoring for system modernization. *IEEE Software*, 35(6):62–67, 2018. doi: 10.1109/MS.2018.4321236.

[20] R. C. Seacord, D. Plakosh, and G. A. Lewis. *Modernizing legacy systems: software technologies, engineering processes, and business practices*. Addison-Wesley Professional, 2003.

[21] S. Stavru, I. Krasteva, and S. Ilieva. Challenges for migrating to the service cloud paradigm: An agile perspective. In *WISE Workshops*, 2011.

[22] S. Stavru, I. Krasteva, and S. Ilieva. Challenges of model-driven modernization - an agile perspective. In *International Conference on Model-Driven Engineering and Software Development*, 2013.

[23] S. Stavru, I. Krasteva, and S. Ilieva. Challenges of model-driven modernization-an agile perspective. In *Modelsward*, pages 219–230, 2013.

[24] M. Strube, S. Runde, H. Figalist, and A. Fay. Risk minimization in modernization projects of plant automation — a knowledge-based approach by means of semantic web technologies. In *ETFA2011*, pages 1–8, 2011. doi: 10.1109/ETFA.2011.6058987.

[25] S. Sundararajan, B. Marath, and P. K. Vijayaraghavan. Variation of risk profile across software life cycle in is outsourcing. *Software Quality Journal*, 27:1563–1582, 2019.

[26] W. Teppe. The arno project: Challenges and experiences in a large-scale industrial software migration project. In *2009 13th European Conference on Software Maintenance and Reengineering*, pages 149–158, 2009. doi: 10.1109/CSMR.2009.64.

[27] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, and D. O'Sullivan. The butterfly methodology: a gateway-free approach for migrating legacy information systems. In *Proceedings. Third IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.97TB100168)*, pages 200–205, 1997. doi: 10.1109/ICECCS.1997.622311.

[28] A. Yamashita. Modernization from a maintenance process perspective: Challenges and lessons learned. *Journal of Advances in Information Technology*, pages 107–113, 01 2017. doi: 10.12720/jait.8.2.107-113.

Comparison of Automated Integration Testing Methods for Component-based Software Systems

Michaela Koval
michaela.koval@rwth-aachen.de

Moritz Brückner
moritz.brueckner@rwth-aachen.de

ABSTRACT

Integration testing is a vital part of the development of component-based software systems as it ensures that individual components of a software interact with each other as expected. Automating the process of generating integration tests helps to achieve good integration test coverage and allows for running integration tests very early in the development life cycle of a software with little effort. This increases the chances of discovering issues before they have an impact, and the software's quality and reliability as a whole is maintained on a high level throughout its entire development. In this paper, we discuss and compare two major approaches for automated integration testing: (i) specification-based methods and (ii) model reconstruction methods. First, we provide a comprehensive view of specification-based methods in integration testing by covering the verification of the functional aspects essential for the components' interactions and the expected temporal correctness. Additionally, we focus on model reconstruction techniques for systems without predefined models or specifications which generate integration test cases by analyzing execution traces and leveraging existing unit tests.

1. INTRODUCTION

The ISO standard on systems and software engineering vocabulary defines integration testing as "testing in which software components, hardware components, or both are combined and tested to evaluate the interaction among them" [2]. While it may seem that successful testing of individual components for correctness implies that these components can be seamlessly combined without causing problems, some errors only become apparent on the integration level [13, 16, 28]. This induces the need for dedicated integration tests to help developers to obtain a reasonable degree of certainty on the correctness of their software, and in fact, integration testing has become a critical part of modern software development. Studies on this topic (cf. [6, 25]) are scarce, presumably due to large amounts of closed-source code on one hand and the difficulty of identifying and qualifying software faults in an automated fashion on the other.

Automated testing [1] is the act of automatically conducting specific tests against a certain system. This is a preferred testing strategy because—given the size and complexity of modern software—writing tests by hand is tedious and error-prone. The automated testing phase, however, is incorporated in the broader concept of *test automation*, which involves the utilization of different tools for the automation of other various aspects of the testing process as well. *Automated test generation*, as fraction of test automation, is a prerequisite for an efficient conduction of the automated testing. It not only contributes to saving valuable development time, but also facilitates larger test case coverage of the code's behavior.

This paper specifically considers integration testing techniques for component-based software systems, which are software systems that are composed of multiple more or less independent components. Component-based software systems naturally encourage separation of concerns [32] and components can be combined, reused and adapted to other systems [23, 32], which is an important aspect in the development of commercial off-the-shelf software.

The structure of this paper is as follows: Section 2 presents various techniques for automatic integration test generation from two major categories, specification-based techniques (Section 2.1) and model reconstruction techniques (Section 2.2). Afterwards, we compare and discuss the presented approaches and their categories in Section 3. In the end, we conclude our work in Section 4.

2. APPROACHES TO AUTOMATED INTEGRATION TESTING

In this chapter, we present a selection of different approaches to automated integration testing that can be found in scientific literature. We have chosen to present several applications of model reconstruction and specification-based testing approaches, as they have complementary strengths and address different aspects of integration testing. While model reconstruction testing methods provide testing based on a global representation of the interaction within a system, the specification-based testing approaches ensure adherence to the specification of the tested system.

FSSE Seminar 2024 RWTH Aachen University, Germany.

2.1 Specification-based Methods

Specification-based testing techniques have promising results for software quality assurance and differ from other techniques as they view the software as a "black-box" and take into account only the software inputs and outputs. These techniques make use of formal languages that adapt mathematical notations for a precise definition of the specification of a system under development. The idea behind specification-based approaches is to verify whether the developed code coincides with the specified behavior.

Common integration testing methods ([27], [21]) that apply specification-based techniques rely on LTL (*Linear Temporal Logic*) representations [10] of the specification. The property-based testing system *Quickstrom* [21] allows its users to specify the behavior of their web applications as a dialect of LTL. Based on that, the web application is treated as a reactive system and is automatically tested, via a static analyzer, against a substantial amount of automatically generated system interactions. Another LTL-based method [27] presents a way to generate a set of test cases, finite in length and in size, from all possible executions of a system, ensuring that the selected test suite is sufficient for testing the presented specification.

On the other side, there are specification-based methods that apply an alternative representation of the specification. The work of Miao and Liu [20] presents an approach for providing a representation of the architecture of a given system that specifies the relations between system operations. This model is constructed based on the predefined expected behavior and the integration tests are generated by extracting all possible execution paths as properties from the model.

The scientific value of specification-based methods applied in automated integration testing originates from the approaches that are used for the automatic generation of the suitable test suite. The actual automatic execution of the tests can be run against the system using the scripts, reflecting the automatically generated test cases, and integrating them within a CI/CD pipeline. Therefore, in this section, we are presenting two specification-based techniques that aim to automatically generate integration tests. The first one is framework that generates an LTL-test-suite by using the counterexample mechanism of a model checker [27]. Furthermore, we present an approach that generates tests from the formal functional scenarios defined by the provided specification in form of *Condition Data Flow Diagrams* [20].

2.1.1 LTL-based testing approach

Using formal methods is proven to be beneficial for supporting the development of software and hardware systems, as they facilitate the formulation of the requirements and provide the foundation for conducting various automatic verification techniques such as model checking [30].

The approach proposed by Lee, Sokolsky and Tan [27] considers a specification-based testing approach in which the system specifications are formally presented in LTL [10]. The level of expressiveness of LTL allows the specification of *safety*, *fairness* and *liveness* properties and therefore the behavioral and temporal requirements to the system or to its components can be captured and formally expressed. In order to verify the derived LTL properties, it has to be checked whether the system satisfies them in all of its possible executions, which can lead to tests that are infinite in both length and size.

The approach focuses on generating finite tests that provide sufficient coverage on the property being tested, under the assumption that a model of the system is provided incorporating the derived LTL properties. By leveraging a model-checking technique, the approach presented in this section could be enhanced into a systematic strategy that aims to automatically generate integration tests.

In order to ensure the existence of a finite test suite, the *property coverage criteria* [4] is applied. Let f be a LTL property that represents the provided specification and ST be a *property-coverage* test suite that mirrors a subset of all possible executions of a system T_s. The property f could be broken down to a finite set of sub-formulas. The idea is to find a test suite ST such that the system T_s passes each one of its tests, e.g. the collection of tests are in accordance to the presented specification, and ST covers every sub-formula of f. A test $t \in ST$ *covers* a sub-formula ψ of f if the system T_s that passes t does not satisfy a *mutation* of the property f. Informally stated, a mutation of f ($f[\psi \leftarrow \phi]$) is an altered version of f in which ψ is replaced by an non-equivalent sub-formula ϕ.

LTL is supported by many established model checkers, such as SPIN [14], and this advantage is used with the aim to generate a test suite w.r.t. the property-coverage criteria. Model checkers [30] are tools used to automatically define whether a certain model satisfies a provided property. When a property is violated, the model checker produces a counterexample, also known as a *witness*, which demonstrates how the system violates the property by providing the trace that lead to the violation. By utilizing the above mentioned feature of the model checkers, the test suite is generated the following way: For each sub-formula ψ of the property f that affects the system T_s, a model checker verifies the *lasso-shaped witness* l of ψ on the provided model of the system T_s.

A *lasso-shaped witness* [24] is a specific witness that exhibits a cyclic behavior. As l is a witness and therefore the model of T_s does not satisfy it, the model checker is expected to produce a counterexample from which test cases can be derived that cover the required behavior of the system.

However, these tests could be infinite in size because of the "cyclic nature" of the lasso-shaped witness. Therefore, with the goal to limit the length of the tests, the *test-truncating strategy* [27] could be applied. Briefly, the idea is to repeat the loop part of the lasso-shaped test a finite number of times until the same states of the the tested execution are encountered for a second time at the same position during the cycle. This way a finite trace of actions can be extracted that replicates the result of the infinite test. Upon receipt of the finite trace, a test that is finite in size can be generated.

2.1.2 Testing based on Condition Data Flow Diagrams

In [20], an integration testing approach is introduced based on the SOFL [17] (Standard Object-Oriented Formal Language) formal specification that is able to precisely define the expected functions of the system operations, e.g. a collection of processes and activities that are necessary for maintaining the expected behavior of the system. Studies show [17] that both precise textual specification and appropriate diagrams for modeling the system architecture are demanded for effective integration testing, therefore SOFL is used as it can specify various aspects of a system.

Each SOFL formal specification [17] is a *hierarchical condition data flow diagram* (CDFD) that is linked with a hierarchy of *module*. Each *module* encapsulates one or more *processes*, which correspond to a certain functionality of the system, *data resources* that are accessed by the process and *invariants* that define the constraints to be conformed by the processes and the data stores. The purpose of the CDFD, in the context of the approach, is to focus on the essential processes that take place in the system operation, ignoring the technical implementation details.

Figure 1 shows the rough structure of the SOFL formulation for a login platform. A SOFL formal specification could be constructed in different levels of condition data flow diagrams that describe the data flow between the processes and the data sources (in the case of Figure 1 there is only one hierarchical level) and constrain the flow control by the data flow of the operational semantics in the form of a textual system description.

The major functions (e.g. processes) of the system shown in Figure 1 are *Credentials Entering* (CE), *Logging In* (LI) and *Exit* (E). These processes are connected by data-flows, which represent the overall function of the module, and each of them consists mainly of input ports, output ports and pre- and post-conditions that describe the expected behavior and data flow. If the input variables satisfy the precondition before the execution of the process, the output variables are required to satisfy the post condition. For instance, process *CE* receives as an input the credentials of a user *user_cred* without a specific precondition, and therefore it is bound to satisfy the post-condition $Post_{CE}(user\_cred)$. During its execution, process CE has a *reading-access* to the *User Data Base* (UDB). The dashed directed line represent *control data flows* that carry no real data, but can cause firings, and the solid lines transmit data that is relevant for the constraints of the linked processes.

Overall, the method for generating integration tests introduced in [20] starts with the assumption that the specification of the system are presented in a SOFL formulation. Consequently, for each of the modules the following steps are executed:

1. *Extract all independent paths directly from the CDFD:*
   A path is an ordered sequence of processes and the data flows connecting them, which starts with a starting process (e.g. its input flow is not an output flow of any other process) and ends with a terminating process (e.g. its output flow is not an input flow of any of the other processes). For instance: The independent paths from the CDFD on Figure 1 are $(used\_cred, CE, user\_id, LI, user\_info)$ and $(user\_id, CE, x, E, exit\_msg)$.

2. *Construct functional scenario forms based on the pre- and post-conditions of each process that is involved in a path:*
   The definition of a *function scenario form* is the following [20]: Given a formal process P in terms of pre- and post-conditions, let the post-condition $Post_P \equiv (C_1 \wedge D_1) \vee ... \vee (C_n \wedge D_n)$, where each C_i is a guard condition, e.g. a condition that contains no output variables, and each D_i is a defining condition that contains at least one output variable. Then each functional scenario form is defined by the conjunction $Pre_P \wedge C_i \wedge D_i$ for all $i \in \{1, ..., n\}$ and Pre_P is the precondition of the process P. In the case of the SOFL specification above, only the process CE has a post-condition and there are no pre-conditions defined, therefore all other functional scenario forms are equal to *true*. Let us assume that the post-condition of CE is $Pre_P \equiv (C_1 \wedge D_1) \vee (C_2 \wedge D_2)$, where C_1 indicates that the user credentials can be found in the user database, C_2 is the negation of C_1, D_1 denotes that the user ID based on the user credentials is output, and D_2 specifies that a signal x is an output. As we have no precondition related to the process CE, as a result, there are two functional scenario forms $(C_1 \wedge D_1)$ and $(C_2 \wedge D_2)$ related to the paths in the module *Login Platform*.

3. *Generate the test cases based on the functional scenario sequences:*
   Each *function scenario sequence* represents one way of how the system is expected to behave. In our simplified example we have only two functional scenario forms in the whole module, therefore we have two functional scenario sequences corresponding to each of the two independent paths. Additionally, these functional scenario sequences are also feasible, as there exists at least one input that satisfies the *testing condition*, e.g. a *testing condition* is each conjunction $Pre_P \wedge C_i$ of the i-th functional scenario $Pre_P \wedge C_i \wedge D_i$ of a formal process P. Based on the found feasible functional scenario sequences, test cases are generated [20].

2.2 Model Reconstruction

Model-based testing methods make use of abstract models that describe the important characteristics of a system under test. Models allow to reduce the complexity of a real system to a common and well-understood (mathematical) representation for which there exist formal techniques to derive, test, and possibly verify its behavior. Such models are often already part of the construction process of a given system [7, 22] and a lot of research papers have been published that focus on generating integration tests from models, often represented by UML state diagrams or UML collaboration diagrams (a good overview of such methods can be found in Section 2 of [3]). Sometimes, however, models (or specifications that can be translated into models) are wrong, incomplete, obsolete or simply do not exist [8, 34]. This is a common issue when working with legacy software systems [34] or commercial off-the-shelf systems [19]. Even if there is a specification, it often does not specify interactions *between* components (such as the order of interface method calls) and instead just accounts for individual components in isolation [34]. Moreover, the software implementation might differ from its specification [32, 29].

To work around these issues, various techniques have been developed which attempt to *reconstruct* a model from a given software system that is expressive enough to aid in the generation of integration tests. While there exists a great amount of scientific research about model reconstruction in general, there are only few publications specifically considering model reconstruction in the context of integration testing. Common approaches make use of execution traces of the system to generate models based on finite state machines [34, 18, 11, 26] or Mealy machines [26], but there exist other techniques such as an approach developed by Wild and Lichter [32] that creates a graph-based model of the system under test based on unit test executions of the

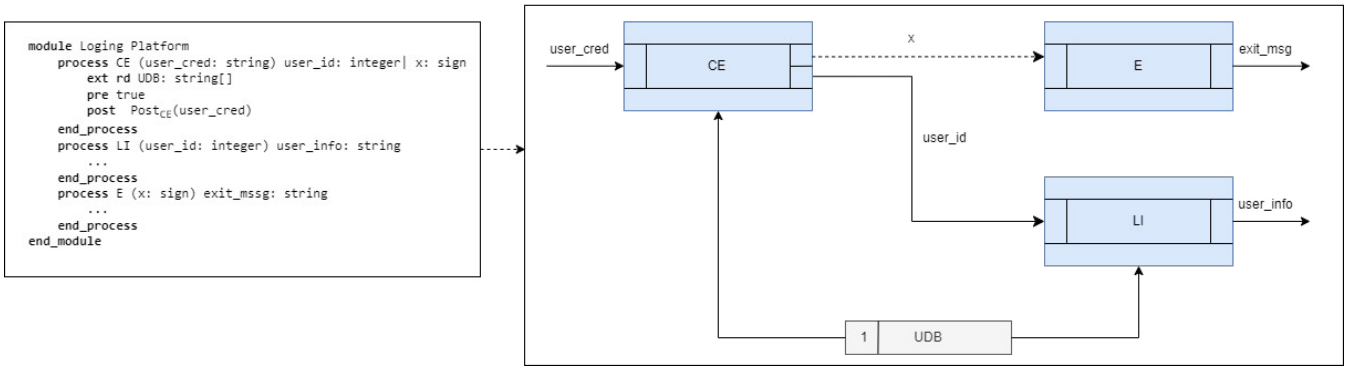

Figure 1: Simple structure of SOFL specification

system. A similar graph-based approach was proposed by Le Traon *et al.* [15], although the authors assume already existing knowledge about class and method dependencies in a system as found in UML class diagrams. Similar to the run-time state extraction used in [34], it is possible to automatically derive (possibly incomplete) class and method dependencies from run-time information if the system under test allows for adequate run-time reflection or instrumentation.

To showcase the methodological differences of various model reconstruction approaches, we present two characteristic techniques that were specifically developed to generate integration tests: (i) the *Substra* framework developed by Yuan and Xie [34] which reconstructs a model from execution traces (Section 2.2.1), and the *UTBI* meta-model (ii) developed by Wild and Lichter [32] which reconstructs a model based on already existing unit tests (Section 2.2.2).

2.2.1 Substra: Reconstructing models from execution traces

Substra [34] is a framework that automatically generates integration test cases for component-based software systems by creating models based on traces collected during executions of the system under test, which contain "executed method calls, object fields' value at entry and exit points of method calls, etc." [34]. Using the execution traces, Substra then creates *object state machines* (OSMs) for all components of interest. These object state machines model possible actions of a single component; the states of the OSMs are the observed states of the component they represent, the transitions of the OSMs are the observed interface method calls of the component they represent. Here, the authors of Substra define the state of components "based on the values of all the fields transitively reachable from the object [i.e., the component in a broader sense; M.B.]" [34], following the *WholeState* technique from Xie *et al.* [33].

Because integration testing is only concerned with interactions between components, Substra only needs to consider *boundary calls*, which are method calls from one component to another component (i.e., the caller is a different component than the callee). From the generated OSMs and a list of boundary calls identified by Substra, the framework generates another state machine for the entire (sub-)system under test, the *subsystem state machine* (SSM). This state machine works similar to OSMs, but (i) instead of considering states of individual components, it considers the union of states of its components, and (ii) instead of using arbitrary method calls as transitions, its transitions are made up from the boundary calls identified earlier. Whenever there is a subsystem state $\{\dots, s_1, \dots, s_2, \dots\}$ for two components in states s_1 and s_2 and there is a single boundary call which sets the state of the first component from s_1 to s_3 and the state from the second component from s_2 to s_4, then the boundary call results in a transition in the SSM from $\{\dots, s_1, \dots, s_2, \dots\}$ to $\{\dots, s_3, \dots, s_4, \dots\}$.

While constructing the SSM, Substra also analyzes for all function calls c_1, c_2 whether the return value c_1 is used as an argument of c_2 and—in the case of object oriented code—whether c_2 is a call of a member function of the return value of c_1. These relationships, which Yuan and Xie call *define-use constraints* [34], are stored as guard conditions on transitions in the SSM for choosing correct parameter values when generating test cases later in the process. The authors of Substra do not specify whether the guard conditions are also used in their proper sense (i.e., to enable or disable certain transitions) [34], but we deem it unlikely since generating state machines based on execution traces should prevent invalid transitions from occurring in the first place.

Once the SSM is constructed, Substra uses it to generate integration test cases for the (sub-)system described by the SSM. To generate a new test case, Substra follows a random path on the SSM (possibly including loops and starting at the system's initial state) for a given maximum sequence length and outputs the sequence of function calls on that path. For all function parameters in a test case, Substra then chooses values based on the following rules:

- For parameters that are part of a define-use constraint, use variable names from the define-use relationships stored with the SSM.
- For parameters that do not belong to a define-use constraint, use random values for primitive types based on a predefined range (*fuzzing*) to also exercise robustness in terms of unexpected or invalid inputs. In case of more complex data types, use values encountered in the execution traces by re-executing method sequences that lead to these values.

Yuan and Xie [34] do not state how their technique automatically differentiates between failed and successful tests

(apart from the obvious fact that some errors will result in program crashes or exceptions), so one has to assume that there still is manual labor involved when working with Substra.

2.2.2 The UTBI model: Deriving component expectations from unit tests

The *Unit Test Based Integration* (UTBI) meta-model [32] is a model of components and their unit tests in a component-based software system with the goal to derive integration tests from already existing unit tests. It is based on the idea that all unit tests have certain *interaction expectations* if they test behavior that causes direct or indirect interactions with other components [32]: A unit test usually stimulates a component under test in some way and in turn expects a correct response to the stimulation which can happen as a result of an interaction with other components. Given that the already existing unit tests are implemented in a way that allows for extraction of information about the unit test cases, their interfaces, and the communication between the interfaces, it is possible to derive a UTBI model and to use it to generate integration tests in an automated fashion by analyzing communication paths in the model. An example implementation of this approach is the software InterACt [31].

Mathematically, the UTBI model is a directed labeled graph whose nodes and edges both maintain (potentially empty) sets of property-value pairs (the graph is a variant of the *property graph model* [5]). Nodes and edges in the UTBI model represent entities of the software components under test and their relations between each other; the type of these entities and relations is described by the graph's labels (e.g., the label `Component` as used in Figure 2 is used for nodes representing software components). In the UTBI model, each component provides incoming and outgoing interfaces that handle the interaction between components and their unit test cases as well as the communication between multiple components. Interfaces follow an API-path-like design, i.e., every interface exhibits a `protocolData` property that defines the functionality of that interface. For example, an interface with the protocol data `/placeOrder` could be called to instruct a component to place a product order (see Figure 2). Interfaces can be *bound* to other interfaces of the opposite direction (incoming resp. outgoing) which represents communication paths between multiple components; an outgoing interface can activate the incoming interfaces bound to it. Moreover, interfaces also maintain a `visibility` property whose values can either be `PUBLIC` (used for interfaces which are meant to be called from outside of the component) or `PRIVATE` (interfaces that are used for internal communication between components). It is assumed that private interfaces are always bound to at least one other interface.

Unit tests are modeled as abstract test cases and concrete test cases. Abstract test cases act as templates for test cases without concrete values and expected results [32]; the actual implementation of a test case is represented by concrete test cases.

The UTBI model represents the actual communication between multiple components or between components and their concrete unit test cases by means of messages with an application-specific payload that form sequences triggered by a unit test case when it is executed. At the beginning of each sequence there is a *stimulus message* that stimulates specific behavior from a component through an incoming interface of that component. The activated component may then send a *component response message* back to the unit test case or to other components, which in the latter case is realized by activating *bound* interfaces. A concrete test case may also expect an *environment response message* which happens as a result of an incoming *component response message* sent by another component. Note that messages are always considered from the perspective of a specific unit test (a unit test only sees its own component, other components are usually replaced by mocks during unit testing); stimulus or environment response messages from a unit test in the UTBI model can be conceptually replaced by component response messages from the view of a different unit test. For instance, in the example illustrated in Figure 2, the interface **I4** is stimulated by the concrete test case **UTC-2** via the message **M4** when the test case is executed, but in a real run of the program, the same interface (or rather its real-world equivalent) might instead be activated by the real-world equivalent of **I2** through a real-world equivalent to the modeled component response message **M2** instead.

UTBI models are first created for each component in isolation before they are merged into a global UTBI model in which bound interfaces are determined in a protocol-specific way. Once the global UTBI model is created, it is possible to check for various kinds of integration faults using *Regular-Path Queries* (RPQs) [12] and their extension *Two-way Regular-Path Queries* (2RPQs) [9] (following [32] and using terminology from Leung and White [16]):

- **Missing/extra function faults:** A component expects another component to have a function which does not exist. This is the case if there is a private interface that is not bound to another one of the opposite direction.

- **Missing/wrong call faults:** The interaction expectation of unit tests can be used to check whether more complex communication paths within a software system are executed as expected. If a unit test expects a response to a message m_1 in the form of another message m_2, one can systematically check whether there exists an interaction path in the application that connects m_1 with m_2 in a way that results in m_2 being sent after m_1. If no such path exists, the unit test exhibits a wrong expectation about the system.

  For the example given in Figure 2, a potential interaction path between the messages **M2** and **M3** could be (**I2**, **I4**, **I5**, **I3**) (cf. [32, Figure 4]).

  However, even if a path exists for a given interaction expectation, it can not be guaranteed that the interaction expectation is actually correctly fulfilled, as the payload of the messages can influence the interaction path and thus also needs to be considered when testing [32].

The presented technique is not able to find *extra call faults* (i.e., functions that are called as part of an interaction even though they should not be called) as the unit tests do not include information about what functions should *not* be called [32]. Likewise, the UTBI model does not consider *wrong function faults* (i.e., functions that do not work as expected) since it is assumed that this kind of fault is found on the unit test level [32].

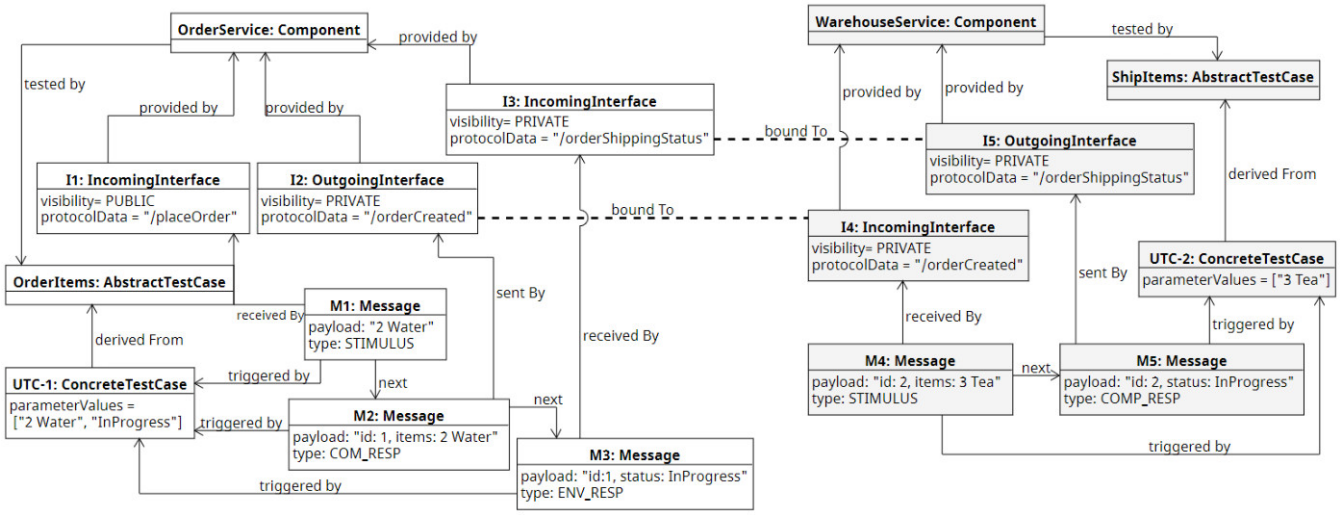

Figure 2: "A property graph of an example UTBI model containing two interacting components with their unit test cases" [32].

The actual integration test cases are generated as follows: First, possible interaction expectations are derived from the UTBI model which are then translated to sequences of abstract unit tests including the interaction paths described by the interaction expectations. Each abstract unit test case in the sequence is then mapped to a corresponding integration test which replaces stimulus and environment response messages with corresponding component response messages from other related unit and integration test cases, that in turn influence the contents of other messages within the integration test. In case of the example illustrated in Figure 2 and a given interaction expectation of **M3** being a response to **M2**, there are two integration test cases generated: One for the **OrderItems** unit test case (**ITC1**) and one for the **ShipItems** test case (**ITC2**). During the transformation of the **ShipItems** test case to **ITC2**, the stimulus message **M4** is replaced by the component response message **M2** which in turn also replaces the message **M5** with a corresponding message **M5'** that contains a new payload computed by the **WarehouseService** component in response to **M2**. Likewise, the environment response message **M3** in **ITC1** is replaced by a corresponding component response message from **ITC2** that used in place of **M5**.

3. COMPARISON OF INTEGRATION TESTING METHODS

Conducting a comprehensive study of many integration testing methods on multiple large and representative code bases would be desirable, but it is, of course, out of scope for this paper and a challenging endeavor since different methods are developed with different use cases and different technologies in mind. This is a huge obstacle for investigating the strengths and weaknesses of individual methods in an unbiased way, as one might need to adapt, modify, or combine methods in order to get them to work on a given code base, potentially distorting the method's functionality as well as the author's intentions. Moreover, authors often do not disclose enough information (such as source code) to completely reproduce their approaches in all their facets, making it difficult to be certain about the correctness of an implementation of an approach. To still provide a broad overview of the strengths and weaknesses of the presented methods, we compare them in terms of their theoretical capabilities as described in the respective papers.

Strengths of model reconstruction techniques.

Model reconstruction techniques come in handy if there is no specification or model for a system under test like it is often the case with third-party components such as commercial off-the-shelf components. They can also counteract common issues with existing models or specifications (such as incompleteness or wrong expectations) as discussed in Chapter 2.2.

Because model reconstruction approaches usually exhibit a high degree of automation and can dynamically adapt to changes in a system, they allow for very early testing.

Weaknesses of model reconstruction techniques.

If the system under test is a black-box, the model can only be inferred from system behavior visible from the outside and thus might be incorrect or incomplete. In case of the Substra framework, for example, if the initial application runs used to generate the execution traces are erroneous or don't cover all of the system's behavior, the generated integration test cases are erroneous or incomplete as well. The UTBI model uses unit tests instead, which will result in invalid or incomplete integration tests if the unit tests themselves are wrong or incomplete.

In addition to that, some model reconstruction methods such as Substra may lead to many generated tests that are duplicates of each other (cf. [34]) since it is possible to (i) include arbitrary detours over unrelated functionality and to (ii) use multiple values exercising the same behavior when generating test cases. However, the severity of this limitation varies between techniques, as some are able to avoid duplicates [19, 18]. In the case of finite state machines, methods are prone to generate redundant test cases if the generated state machines are not *minimal* [11].

As with all fully automated testing methodologies, there is a need for a test oracle (i.e., a source of truth that decides

whether a test is successful or not), which—because there is no specification or model—must be inferred from the software system itself and thus might be incorrect or incomplete as well.

A different challenge for model reconstruction techniques is the implemented design or architecture of the software system: The software must be written in a way that enables the test case generation methods to infer a model from it. The UTBI model, for example, needs unit tests to follow an API-path-like format from which it is possible to derive the required information. If it is not possible to automatically construct messages from the given unit tests that match the format required by the UTBI model, it cannot be applied without additional effort.

Similarly, not all components might expose enough information to allow inferring certain models. While the UTBI model is relatively technology-agnostic by treating the system under test like a black-box, the Substra framework requires run-time reflection on the *state* of the system, which in practice might not be available.

Strengths of specification-based techniques.

In comparison to the model reconstruction testing techniques, where models are used for the representation of the expected behavior, by specification-based techniques a system is being tested against the specification and the requirements of the system under test. Therefore, more of the expected functionality could be tested and it ensures that the code meets all of the necessary requirements.

Additionally, the specification-based methods tend to lead to a more efficient testing, since it is based on well defined set of rules that represent the expected system behavior, i.e. by the CDFD-approach (Section 2.1.2) these set of rules mostly covers the pre- and post-conditions of the process. Thus, the testing could contribute to detecting bug that go undetected by the application of a not so specification-driven technique.

Specification-based technique also uses the formal specification of a system to derive test cases from, which could be observed by the both specification-driven approaches presented in the current work (Section 2.1.1 and Section 2.1.2). Thereby, the focus is on representative test cases that provide the required coverage and on minimizing duplicate testing, which increases performance.

Weaknesses of specification-based techniques.

One of the biggest limitations of specification-based approaches is its depending on an accurate, comprehensive and consistent specification, which could be challenging, expensive and difficult to maintain. Furthermore it could be limiting of terms of testing of unexpected behavior, as the specification-based testing is executed only towards an established criteria. Therefore, it is mainly suitable for the testing of the functional requirements of a system, not so much for the usability-related requirements.

4. CONCLUSION

In this paper, we presented and compared different approaches to automatic integration testing with a focus on specification-based techniques and model reconstruction. We summarized the current state of research in this specific area and illustrated its difficulties as well as the advantages and limitations of the presented integration testing techniques, which each find a specific niche of application. Due to the scope of this paper, only a rough comparison could be made. Still, this knowledge could support practitioners to select the most appropriate methods for their specific needs or even to apply a combination of the described strategies. Therefore, the current work could contribute to a more reliable integration testing process in future efforts.

Aspect	Spec.-based Methods	Model Reconstr.
System requirements	System spec. needed	No model or spec. needed, but other method-specific information
Automated	Yes	Yes
Early testing	Possible	Possible
Completeness	Test all requirements	Depends on system
Correctness	Need correct spec.	Depends on system
Duplicate tests	Minimized	Common

Table 1: Overview of our results

5. REFERENCES

[1] ISO/IEC/IEEE 29119-1:2013(E): Software and systems engineering – Software testing, 2013.

[2] ISO/IEC/IEEE 24765:2017(E): Systems and software engineering – Vocabulary, 2017.

[3] S. Ali, L. C. Briand, M. J.-u. Rehman, H. Asghar, M. Z. Z. Iqbal, and A. Nadeem. A state-based approach to integration testing based on UML models. *Information and Software Technology*, 49(11-12):1087–1106, 2007.

[4] P. E. Ammann and P. E. Black. A Specification-Based Coverage Metric to Evaluate Test Sets. In *Proceedings 4th IEEE International Symposium on High-Assurance Systems Engineering*, pages 239–248, Washington, D.C., USA, 1999. IEEE.

[5] R. Angles. The Property Graph Database Model. In D. Olteanu and B. Poblete, editors, *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 2100 of *CEUR Workshop Proceedings*, Cali, Colombia, 2018. CEUR-WS.org.

[6] V. R. Basili and B. T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42–52, 1984.

[7] A. Bertolino. Software Testing Research: Achievements, Challenges, Dreams. In L. C. Briand and A. L. Wolf, editors, *Future of Software Engineering (FOSE '07)*, pages 85–103, Minneapolis, MN, USA, 2007. IEEE.

[8] M. Bures, K. Frajtak, and B. S. Ahmed. Tapir: Automation Support of Exploratory Testing Using Model Reconstruction of the System Under Test. *IEEE Transactions on Reliability*, 67(2):557–580, 2018.

[9] D. Calvanese, M. Y. Vardi, G. De Giacomo, and M. Lenzerini. View-Based Query Processing for Regular Path Queries with Inverse. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems - PODS '00*, pages 58–66, Dallas, TX, USA, 2000. ACM Press.

[10] E. M. Clarke and I. A. Draghicescu. Expressibility results for linear-time and branching-time logics. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume

354 of *Lecture Notes in Computer Science*, pages 428–437, Berlin, Heidelberg, 1989. Springer.

[11] J. E. Cook and A. L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[12] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. *ACM SIGMOD Record*, 16(3):323–330, 1987.

[13] M. J. Harrold and M. L. Soffa. Selecting and Using Data for Integration Testing. *IEEE Software*, 8(2):58–65, 1991.

[14] G. J. Holzmann. The Model Checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[15] Y. Le Traon, T. Jéron, J.-M. Jézéquel, and P. Morel. Efficient Object-Oriented Integration and Regression Testing. *IEEE Transactions on Reliability*, 49(1):12–25, 2000.

[16] H. K. N. Leung and L. White. A Study of Integration Testing and Software Regression at the Integration Level. In *Proceedings. Conference on Software Maintenance 1990*, pages 290–301, San Diego, CA, USA, 1990. IEEE.

[17] S. Liu, A. J. Offutt, C. Ho-Stuart, Y. Sun, and M. Ohba. SOFL: A Formal Engineering Methodology for Industrial Applications. *IEEE Transactions on Software Engineering*, 24(1):24–45, 1998.

[18] L. Mariani, F. Pastore, and M. Pezze. Dynamic Analysis for Diagnosing Integration Faults. *IEEE Transactions on Software Engineering*, 37(4):486–508, 2011.

[19] L. Mariani and M. Pezze. Dynamic Detection of COTS Component Incompatibility. *IEEE Software*, 24(5):76–85, 2007.

[20] W. Miao and S. Liu. A Formal Specification-Based Integration Testing Approach. In S. Liu, editor, *Structured Object-Oriented Formal Language and Method*, pages 26–43, Berlin, Heidelberg, 2013. Springer.

[21] L. O'Connor and O. Wickström. Quickstrom: Property Based Acceptance Testing with LTL Specifications. Preprint (arXiv:2203.11532), 2022.

[22] A. Pretschner and J. Philipps. 10 Methodological Issues in Model-Based Testing. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors, *Model-Based Testing of Reactive Systems*, volume 3472, pages 281–291. Springer, Berlin, Heidelberg, 2005.

[23] C. R. Rocha and E. Martins. A strategy to improve component testability without source code. In S. Beydeda, V. Gruhn, J. Mayer, R. Reussner, and F. Schweiggert, editors, *Testing of component-based systems and software quality: Net.ObjectDays Workshops on Testing of Component-Based Systems (TECOS 2004) and Software Quality (SOQUA 2004)*, volume 58 of *GI-Edition Proceedings*, pages 47–62, Erfurt, Germany, 2004. Gesellschaft für Informatik.

[24] V. Schuppan and A. Biere. Efficient reduction of finite state model checking to reachability analysis. *International Journal on Software Tools for Technology Transfer*, 5:185–204, 2004.

[25] H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge. Programmers' Build Errors: A Case Study (at Google). In *Proceedings of the 36th International Conference on Software Engineering*, pages 724–734, Hyderabad, India, 2014. ACM.

[26] M. Shahbaz. *Reverse Engineering Enhanced State Models of Black Box Software Components to support Integration Testing*. Ph.D. thesis, Institut Polytechnique de Grenoble, Grenoble, France, 2008.

[27] L. Tan, O. Sokolsky, and I. Lee. Specification-based Testing with Linear Temporal Logic. In A. M. Memon, editor, *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration*, pages 493–498, Las Vegas, NV, USA, 2004. IEEE.

[28] J. Tretmans. Model Based Testing with Labelled Transition Systems. In R. M. Hierons, J. P. Bowen, and M. Harman, editors, *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer, Berlin, Heidelberg, 2008.

[29] E. Werner and J. Grabowski. Model Reconstruction: Mining Test Cases. In *Third International Conference on Advances in System Testing and Validation Lifecycle (VALID 2011)*, pages 97–102, Barcelona, Spain, 2011. IARIA XPS Press.

[30] S. Wieczorek, V. Kozyura, A. Roth, M. Leuschel, J. Bendisposto, D. Plagge, and I. Schieferdecker. Applying Model Checking to Generate Model-Based Integration Tests from Choreography Models. In M. Núñez, P. Baker, and M. G. Merayo, editors, *Testing of Software and Communication Systems*, pages 179–194. Springer Berlin Heidelberg, 2009.

[31] N. Wild and H. Lichter. InterACt: a Tool for Unit Test Based Integration of Component-based Software Systems. In *ICSEA 2023: The Eighteenth International Conference on Software Engineering Advances*, pages 58–63, Valencia, Spain, 2023. IARIA.

[32] N. Wild and H. Lichter. Unit Test Based Component Integration Testing. In *30th Asia-Pacific Software Engineering Conference (APSEC 2023)*, pages 1–10, Seoul, Korea, Republic of, 2023. IEEE.

[33] T. Xie, D. Marinov, and D. Notkin. Rostra: a Framework for Detecting Redundant Object-Oriented Unit Tests. In *Proceedings. 19th International Conference on Automated Software Engineering, 2004.*, pages 196–205, Linz, Austria, 2004. IEEE.

[34] H. Yuan and T. Xie. Substra: a Framework for Automatic Generation of Integration Tests. In *Proceedings of the 2006 International Workshop on Automation of Software Test*, pages 64–70, New York, NY, USA, 2006. Association for Computing Machinery.

Documentation in ML-enabled Software Systems: Needs and Challenges

Baran Tanriverdi
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
baran.tanriverdi@rwth-aachen.de

Obaid Ur Rehman
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
obaid.rehman@rwth-aachen.de

ABSTRACT

ML-enabled systems, due to their complexity and dynamic nature, present unique documentation challenges that differ significantly from traditional software systems. This paper conducts a systematic review of the existing literature to identify and address these challenges. We explore the specific documentation needs of various stakeholders, including data scientists, ML engineers, and project managers, highlighting the critical role of comprehensive and up-to-date documentation in enhancing transparency, reproducibility, and maintainability. By examining current documentation strategies, we provide insights into improving clarity and usability. FactSheets and Model Cards enhance transparency and accountability, architectural diagrams offer detailed visual representations of system components, and interactive notebooks integrate code with narrative text and visualizations to support comprehensive documentation. Our findings discuss the existing challenges and aim to support the creation of more effective documentation frameworks in rapidly evolving ML environments, ultimately promoting better communication, accountability, and efficiency across ML projects. Through this review, we offer practical recommendations for improving documentation practices and highlight areas for future research to further enhance the documentation of ML systems.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Software Management—*productivity, programming teams, software configuration management*; D.2.7 [**Distribution, Maintenance, and Enhancement**]: [Documentation, Performance Measures]; I.2.6 [**Artificial Intelligence**]: [Learning, Concept learning, Knowledge acquisition]

Keywords

Machine Learning Documentation, Systematic Review, Documentation Challenges, FactSheets, Model Cards, Architectural Diagrams, Interactive Notebooks, Transparency, Reproducibility, Maintainability, Data Scientists, ML Engineers, Project Managers, Documentation Strategies, ML Systems

SWC Seminar 2024 RWTH Aachen University, Germany.

1. INTRODUCTION

Machine learning (ML) is a transformative technology, that revolutionizes various industries by enabling systems to learn from data and improve continuously. However, the integration of ML into software systems introduces unique documentation challenges, distinguishing it from traditional software engineering. Effective documentation is necessary to ensure the transparency, efficiency, reproducibility, and maintainability of ML models. Unlike conventional software, ML systems involve complex data handling, model training, and evolving algorithms, requiring a specialized documentation approach [3, 8].

The complexity of ML systems requires documentation that not only describes the final product but also captures the processes and decisions involved in model development. This includes data preprocessing, model architectures, training procedures, and evaluation metrics. Proper documentation supports accountability by allowing stakeholders to track model performance, identify biases, and consider ethical implications [3]. Due to the dynamic nature of ML systems, documentation must be continuously updated to reflect changes in data, algorithms, and model parameters [7]. Consequently, comprehensive documentation can mitigate technical debt and improve long-term maintainability of ML systems [30].

Despite the critical importance of documentation in ML projects, current practices often fall short [22]. A recent study from 2024 found that only 39.62% of models and 28.48% of datasets in one of the largest model stores, Hugging Face, have documentation [24]. Frequent updates and rapid iteration cycles complicate maintaining comprehensive and accessible documentation. Traditional documentation tools often fail to capture the complexity and evolving nature of ML workflows. This gap presents significant challenges for data scientists, ML engineers, and project managers, who rely on accurate and up-to-date documentation to perform their roles effectively [13]. Existing solutions lack the adaptability required to keep up with the pace of advancements in ML technologies [21].

Through a systematic review of the literature, this paper will address the following research questions:

- **RQ1:** What are the specific documentation needs of various roles within ML projects?

- **RQ2:** What are the current state-of-the-art solutions for documenting ML systems?
- **RQ3:** What are the main challenges in the documentation of ML systems?

This paper aims to address the aforementioned challenges by conducting a review of documentation practices tailored to the specific needs of different roles within ML projects. By identifying the unique documentation requirements and obstacles faced by various stakeholders, this research seeks to review the usability and comprehensibility of ML documentation. Additionally, it will explore strategies for creating adaptable documentation that can evolve alongside ML systems and review tools and frameworks that facilitate continuous and automated documentation updates [16]. By answering these questions, this research aims to provide insights and recommendations for improving documentation practices in ML-enabled software systems, ultimately supporting better communication, accountability, and efficiency across ML projects.

2. METHODOLOGY

This section outlines the methodological approach adopted for this study, detailing the systematic review process, PICOC framework, search strategy, data extraction and analysis procedures. We used Parsifal, a web-based tool for systematic reviews[1], to support the entire review process and ensure a structured and efficient workflow.

2.1 Systematic Review Approach

To comprehensively examine the documentation needs and challenges associated with ML-enabled software systems, we employed a systematic review approach. Systematic reviews are rigorous methods used to gather, critically evaluate, and synthesize research findings on a particular topic [29]. This approach ensures thoroughness and minimizes bias, providing a solid basis for identifying key themes and gaps in the literature.

The methodological framework guiding this systematic review involved several key steps. Initially, we defined the research questions to direct the review process. Following this, a detailed review protocol was established, outlining the criteria for study selection. An extensive literature search was conducted in several databases to identify relevant studies. The identified studies were then screened using predefined inclusion and exclusion criteria. Key information was systematically extracted from the selected studies, which was then analyzed and synthesized to draw meaningful conclusions.

2.2 PICOC Framework

We applied the PICOC framework to systematically structure our review, defining the Population, Intervention, Comparison, Outcome, and Context [29]. This framework helps in clearly specifying the key components of our review and ensures a focused approach:

- **Population:** This component specifies the groups involved in the study. In our review, the population includes developers, data scientists, project managers, and ML engineers who interact with ML-enabled systems and their documentation.
- **Intervention:** This refers to the treatment or action being studied. For our review, the intervention is the documentation practices in ML-enabled systems, which include various strategies and tools.
- **Comparison:** The comparison component involves comparing the intervention against another treatment or the absence of treatment. We do not have a direct comparison group, as we focus solely on the intervention of documentation practices.
- **Outcome:** This defines the effects or results that are being measured. Our review aims to assess the effectiveness of different documentation methods, the challenges faced in ML documentation, and the improvements in project outcomes resulting from enhanced documentation practices.
- **Context:** This component describes the environment in which the study is conducted. For our review, the context includes both commercial software development and academic environments where ML-enabled systems are developed and maintained.

2.3 Search Strategy

A critical component of our systematic review was the search strategy employed to identify relevant studies. This involved conducting comprehensive searches across multiple academic databases, including Google Scholar, IEEE Xplore, the ACM Digital Library, and arXiv. By using a wide array of sources, we aimed to capture a broad spectrum of research related to ML documentation practices and tools. The search terms were carefully chosen to encompass various aspects of documentation in ML systems, including:

- "Automatic documentation of machine learning systems"
- "ML documentation tools"
- "Documentation practices in ML"

To maintain the relevance and quality of the studies included in our review, we applied specific inclusion and exclusion criteria. These criteria helped us focus on the most relevant and high-quality research. The inclusion criteria were publications from 2015 to 2024, ensuring the review covered recent advancements and trends; studies that specifically address documentation practices, tools, and challenges in ML systems; and peer-reviewed journal articles, conference papers, and high-quality grey literature that provide substantial contributions to the topic. The exclusion criteria included studies irrelevant to ML documentation, such as those focused solely on traditional software systems without an ML component, and outdated practices that do not reflect current trends and technologies in ML documentation. This strategic approach ensured that we gathered a comprehensive set of studies relevant to our research questions.

[1]The entire review process was conducted using Parsifal, a web-based tool for systematic reviews (`https://parsif.al/`).

Figure 1: Overview of the paper selection flow

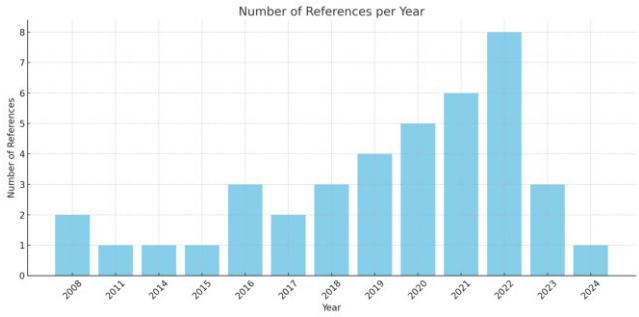

Figure 2: Distribution of references per year

2.4 Data Extraction and Analysis

After selecting the relevant studies, we proceeded with a detailed analysis phase. We scanned the remaining papers thoroughly to decide which were not needed. Additionally, we manually added selected papers not directly in the ML field or published before 2015 if considered relevant. Following these steps, we conducted the data extraction and analysis phase using Parsifal. The data extraction process involved capturing detailed information from each study, including study objectives, methodologies, findings, and relevance to the research questions. The extracted data were organized into categories corresponding to the documentation needs and challenges identified in our research questions. We then integrated the findings across studies, identifying common themes, gaps, and potential solutions for improving documentation practices in ML systems.

Initially, we identified 170 papers through an extensive search across multiple databases (see Figure 1). After applying our inclusion and exclusion criteria, 132 papers remained for detailed analysis. Following the detailed analysis, we finally selected 39 papers for this study. Figure 2 illustrates the distribution of references per year. This figure shows the trend of publication years for the selected references, highlighting the growing interest in ML documentation practices in recent years. This rigorous process ensured that our review was comprehensive and focused on the most relevant studies.

3. RESULTS

Our review of documentation practices in ML-enabled systems highlights a varied landscape where documentation is crucial for ensuring transparency, reproducibility, and collaboration. These findings emphasize the need for customized documentation strategies that meet the specific requirements of different roles within ML projects. By examining the documentation needs of various stakeholders, we can pinpoint best practices and identify areas that require enhancement.

3.1 Role-Specific Documentation Needs

Various roles within ML projects have unique documentation requirements. Understanding these role-specific needs helps in creating documentation that is both useful and efficient. This study focuses on data scientists, ML engineers, and project managers, as they represent the most relevant roles within ML projects. This classification, commonly used in other scientific works, reflects the diverse but interconnected responsibilities essential for the development, deployment, and management of ML systems. This section addresses the first research question (RQ1), exploring the specific documentation needs of these roles. Table 1 summarizes stakeholders, their primary interests, and the common artifacts used within ML-enabled systems.

Table 1: Stakeholder Interests and Artifacts

Stakeholder	Primary Interests	Common Artifacts
Data Scientists	Understand data Understand models Model performance Ensure reproducibility	Data Cards Model Cards Experiment records Evaluation metrics Visualizations
ML Engineers	Understand models Understand data Deployment Maintenance System architecture APIs and data pipelines	Model Cards Data Cards Architectural diagrams Process guidelines
Project Managers	Project overview Timelines Resource management Business objectives	Gantt charts Status reports Risk management plans User stories

3.1.1 Data Scientists

Data scientists require comprehensive documentation to ensure the reproducibility of experiments and the clarity and transparency of their analytical processes. Therefore, it is necessary to document the steps taken during the model development and performance evaluation, including preprocessing, feature engineering, model tuning, and the definition of metrics. This ensures that experiments can be replicated [32]. Maintaining thorough records of experiments, including hypotheses, test results, and conclusions, is a best practice for developers. This includes detailed descriptions of data sources, datasets, and variables, which promote consistency and clarity in data interpretation [15, 27]. Furthermore, summarized data findings and visualizations are vital for communicating insights to stakeholders, thereby enabling informed decision-making [9, 16, 27, 31, 36, 37].

Data scientists face challenges in documenting the extensive details of their work, including maintaining up-to-date documentation, and ensuring it is useful for others. Current documentation approaches are often ad hoc and insufficiently detailed, which can hinder effective collaboration with ML engineers and project managers [16]. Therefore, providing clear documentation of requirements and data quality is important to align team efforts and ensure the smooth integration and operation of ML models [22].

3.1.2 ML Engineers

ML engineers focus on the technical documentation necessary for deploying and maintaining ML systems [27]. Detailed documentation of the system architecture, APIs, and data pipelines ensures that each component is well-defined

and integrated [15, 27]. This foundational information helps developers understand the system's operational aspects and how various technologies interact within it. In addition, ML engineers must possess a depth of knowledge regarding the model design, algorithms, features, training processes, and evaluation metrics [22, 37]. Moreover, comprehensive guidelines on coding standards, testing frameworks, deployment procedures, and environment and dependency information are crucial for maintaining code quality and reliability across different stages of development and deployment [16, 36, 27, 32]. The documentation of model performance in a variety of scenarios enables developers to identify potential weaknesses and implement improvements, thereby maintaining high-performance standards [22, 37].

ML engineers encounter difficulties with integrating ML models into production systems, particularly regarding the technical complexity and the need for clear documentation of model specifications and deployment processes. Their interdisciplinary work with data scientists requires careful coordination and clear documentation to ensure smooth integration and operation [22]. Effective collaboration with data scientists and project managers involves identifying and decomposing requirements and negotiating data quality and quantity to meet project needs [22].

3.1.3 *Project Managers*

Project managers need documentation that provides a comprehensive overview of the project and ensures alignment with business objectives. This includes detailed project roadmaps outlining timelines, milestones, and deliverables to ensure team alignment [31, 34]. Market research insights inform the development of detailed product features and user stories, thereby ensuring that developers and designers have a clear understanding of project requirements [9, 10, 23]. Furthermore, performance reports, which track metrics and analytics related to product performance and user engagement provide crucial feedback for assessing success and identifying areas for improvement [6]. Compliance documents that detail the necessary regulations and standards ensure that the product adheres to relevant legal and regulatory requirements [11].

The key for project managers is to balance various stakeholders' demands while ensuring that documentation is neither too cumbersome for the team nor too superficial to be useful. They must guarantee that documentation practices facilitate the project's overall management without becoming a bottleneck [31]. Managers actively engage in regular communication with data scientists and ML engineers, facilitating the exchange of information through meetings and updates. Understanding the basics of ML solutions helps project managers to coordinate effectively with technical teams and ensure the integration of technical and business objectives. Effective collaboration is central to ensuring that efforts are aligned, work is seamlessly integrated, and that data quality and project requirements are met [22].

3.2 State of Art of Current Solutions

In the field of ML, which is undergoing rapid evolution, the need for robust documentation methods is becoming increasingly significant. A variety of tools and methodologies have been developed to ensure transparency, accountability, and reliability in ML systems. Among these, Factsheets, Model Cards, Data Cards, architectural diagrams, and interactive notebooks have emerged as instruments for documenting ML systems. In order to address the second research question (RQ2), we have decided to focus on the aforementioned tools, which we see as the most relevant in current practice. This section reviews these current solutions, highlighting their structure, implementation, and impact on the ML field.

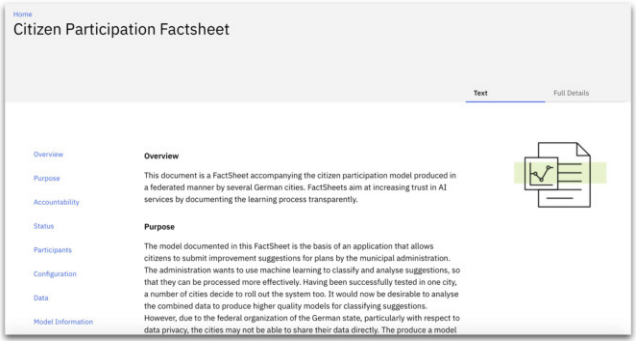

Figure 3: Exemplary FactSheet [5].

3.2.1 *FactSheets*

FactSheets are a valuable tool for documenting ML systems, providing structured information about the development, deployment, and operational attributes of these systems (see Figure 3). They serve as a form of transparent reporting, aimed at increasing trust and accountability in ML systems. FactSheets document essential details about ML models and services, addressing various dimensions of the AI model's lifecycle such as data, performance, safety, reliability, security, and lineage [28]. By standardizing the communication of these attributes, FactSheets help stakeholders, including developers, users, and regulators, to make informed decisions about the ML systems they interact with [4, 5].

The methodology for creating FactSheets involves several iterative steps that include key roles such as project managers, data scientists, and ML engineers. Initially, it involves understanding the needs and information requirements of potential FactSheet consumers. This step includes gathering insights through interviews, surveys, and feedback sessions to determine what information is crucial for different roles and how it influences their decision-making processes. Next, it is important to identify the types of information that FactSheet producers can provide. Data scientists typically document model creation processes, including data preprocessing, model training details, and performance metrics. ML engineers contribute information on testing procedures, deployment strategies, and monitoring details. Project managers provide project goals, timelines, and accountability structures. The FactSheet template should be adaptable to accommodate different types of information and formats, ensuring it meets stakeholders' diverse needs [4]. Filling in the FactSheet template with actual data from the AI model or service involves multiple iterations and refinements. Feedback from both producers and consumers of FactSheets is essential to ensure the captured information is accurate, understandable, and useful [28].

FactSheets have been successfully applied in various domains to document both standalone models and complex

AI services, facilitating their integration into larger systems. For instance, FactSheets have been tailored to document the transparency and traceability of analog in-memory computing AI models, highlighting their applicability across different hardware and software contexts [22, 28].

Despite the robust framework offered by FactSheets, their creation is exhaustive due to multiple iterations, and their implementation faces challenges such as the dynamic nature of AI models that evolve over time. Future research aims to refine FactSheet templates and methodologies to ensure they remain adaptable and comprehensive. Additionally, exploring the integration of FactSheets with other transparency tools like Model Cards and Datasheets for Datasets could further enhance their utility and adoption [13, 19].

Datasheets and Data Cards provide broad insights into datasets and models, enhancing transparency and facilitating responsible use. Datasheets for Datasets aim to communicate the details of datasets, including the data collection process, the composition of the dataset, and potential biases or limitations. This descriptive documentation ensures that users understand the origins and characteristics of the data, which is essential for evaluating its suitability for specific applications and for identifying any inherent biases that could impact model performance [14]. Similarly, Data Cards describe datasets from multiple perspectives, including data view, collection process, composition, intended use, and quality, further promoting transparency and accountability in ML systems [25].

3.2.2 *Model Cards*

Model Cards have as well emerged as a significant tool for documenting ML models. While FactSheets provide comprehensive broad documentation for entire AI systems, Model Cards focus specifically on the performance and evaluation of individual ML models. As shown in Figure 4, the primary objective of Model Cards is to enhance transparency and accountability in ML by providing detailed documentation of a model's performance, intended use cases, and limitations across different conditions and demographic groups [4]. Therefore, this tool aims to ensure that users of ML models understand the conditions under which a model performs well, its potential biases, and its limitations. This is crucial in preventing the misuse of models in inappropriate contexts and ensuring that they are applied in ways that align with their design and training [19].

A typical Model Card includes several key components:

- **Model Details**: Information about the model's architecture, training data, and intended use. This section also outlines the version of the model and any relevant licensing information.
- **Intended Use**: Specifies the applications for which the model is designed and the contexts in which it should not be used.
- **Factors**: Lists relevant demographic or phenotypic factors (e.g., age, gender, race) and intersectional groups that are considered in the model's evaluation.
- **Metrics**: Provides benchmarked performance metrics across different conditions and groups to highlight any disparities or biases in the model's behavior.

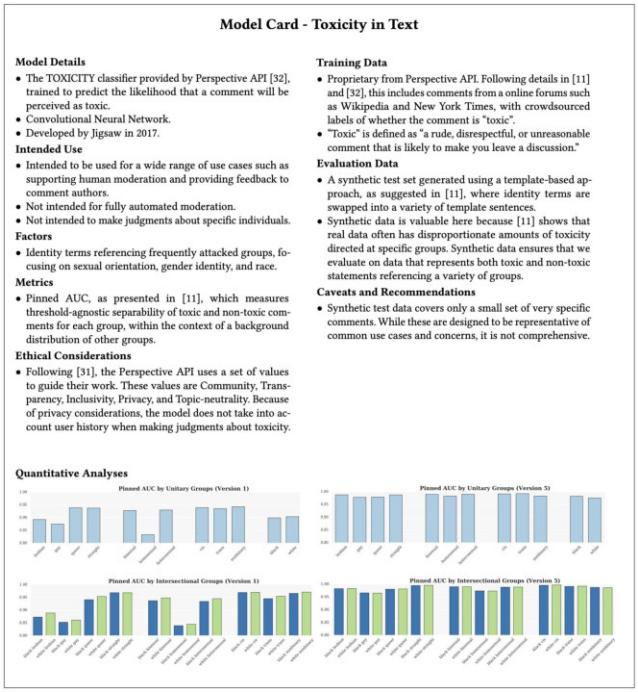

Figure 4: Exemplary Model Card [19].

- **Evaluation and Training Data**: Describes the datasets and methodologies used for evaluating and training the model.
- **Ethical Considerations**: Discusses potential ethical issues related to the model's deployment, including fairness, accountability, and transparency.
- **Caveats and Recommendations**: Offers guidance on the limitations of the model and recommendations for its appropriate use.

The implementation of Model Cards involves a systematic approach to documenting and reporting ML models. For instance, Google has developed a Model Card Toolkit (MCT)[2] to streamline the creation of Model Cards. The MCT provides a schema in JSON format for capturing essential details about a model and includes tools for visualizing this information in a standardized format. The toolkit supports the integration of Model Cards into the development workflow, making it easier for developers to create and maintain documentation.

Despite their benefits, the adoption of Model Cards also presents challenges. Creating comprehensive and accurate Model Cards requires significant effort and expertise. Additionally, there is still a need for standardized methodologies for evaluating and reporting model performance, particularly for complex and high-stakes applications [35]. Future research should focus on developing robust frameworks and tools to support the creation, automated updating and sharing of Model Cards, as well as exploring ways to integrate community feedback into the documentation process [19].

[2]https://blog.research.google/2020/07/introducing-model-card-toolkit-for.html

3.2.3 Architectural Diagrams

Architectural diagrams visualize the structure and interactions within ML systems. These diagrams provide a big-picture view that helps everyone involved to understand how the system is set up and how its parts work together. They make it easier to see the overall design and the connections between components [18, 38].

Various types of architectural diagrams serve distinct purposes in documenting ML systems. A system overview diagram provides a high-level perspective of the entire ML system, highlighting key components and their interactions without looking into details. Component diagrams, on the other hand, detail the organization and dependencies among various components, focusing on software modules, their roles, and how they work together to deliver functionality [33]. Data flow diagrams illustrate the movement of data within the system, detailing processes, data stores, and external entities, which is crucial for understanding data processing and transformation steps [39]. Deployment architecture diagrams show the physical distribution of software components across different nodes such as servers and databases, visualizing their distribution and execution in the deployment environment [33]. Sequence diagrams capture the sequence of interactions between components or objects in specific scenarios, highlighting the flow of messages and the order of actions to demonstrate the dynamic behavior of the system [1]. Interaction diagrams focus on the interactions between components during specific operations, often including sequence diagrams or flowcharts to depict the order of operations and dependencies [18]. Security architecture diagrams address the security measures within the ML system, covering aspects like privacy, integrity, and access control to ensure security concerns are adequately managed.

Architectural diagrams make it easier for developers to keep their documentation up-to-date and understandable [18]. However, there are challenges, such as the time and expertise required to create detailed diagrams and the lack of standardized methods for complex systems [2]. Additionally, it is difficult to capture the dynamic aspects of many ML components [33, 38]. Future efforts should focus on developing better tools and frameworks to make this process more efficient and automated, and to enable teams to collaborate effectively on these diagrams [2].

3.2.4 Jupyter Notebooks

Interactive notebooks, such as Jupyter Notebooks, integrate executable code with explanatory text and visualizations. This integration supports a literate programming approach, presenting the logic underlying the code alongside the code. Consequently, notebooks facilitate the usability and comprehensibility of ML system workflows by providing a transparent, cohesive and reproducible documentation format [17].

Data scientist workflows frequently entail iterative and exploratory processes. Notebooks support this process by allowing users to execute code in small, manageable blocks (cells), thus making it possible to test and refine models incrementally. This iterative process is well-documented within the notebook, providing a clear record of the analysis steps, and decision points, and ensuring reproducibility, which are essential for maintaining the reliability and integrity of ML systems [12]. The interactive nature of notebooks also enables collaboration among data scientists and ML engineers. Notebooks can be shared, reviewed, edited and documented by multiple users [12].

Although notebooks offer several advantages, they can become disorganized with duplicated, out-of-order code, and minimal documentation, making them difficult to maintain and understand [17]. To address these challenges, various tools have been developed to improve the quality and maintainability of notebook documentation. These tools enforce best practices for code quality, reproducibility, and documentation, ensuring that notebooks remain well-structured and accessible [26]. In addition, automated documentation tools analyze the code within notebooks in real time, generating up-to-date documentation that aligns with the latest code changes. For example, techniques for labeling notebook cells with ML pipeline stages and generating dependency graphs further aid in navigation and understanding [17]. Such tools are essential for maintaining clear and comprehensive documentation as ML projects growing in complexity struggle to keep their documentation updated [20].

Furthermore, as ML models and datasets grow in size, executing notebooks on local machines becomes impractical. Integrating notebooks with high-performance computing (HPC) systems addresses the limitations of scalability, facilitating distributed training and hyperparameter optimization for large datasets and complex models. The integration of interactive widgets and real time monitoring within the context of notebooks enhances human-in-the-loop interactivity, which is a key aspect in the effective development of ML models [12].

3.3 Discussion

Our analysis revealed diverse documentation needs within ML projects, necessitating tailored practices to effectively support all stakeholders. We examined the most relevant tools currently used for documenting ML systems and identified several significant challenges, addressing the third research question (RQ3).

A primary challenge is the inherent complexity of ML systems. Current documentation practices and tools are often insufficient for ML needs. Traditional documentation tools typically fail to capture the complexity of ML workflows, resulting in inconsistent and incomplete records. Given processes require detailed documentation to ensure reproducibility and clarity. Capturing the rationale behind model choices and specific configurations is essential, but challenging. In addition, the dynamic nature of ML systems, which are frequently updated with new data and evolving algorithms, complicates the task of maintaining up-to-date documentation. The rapid iteration cycles characteristic of ML projects can quickly render existing documentation obsolete.

Effective collaboration and communication are important to success of ML projects, but the interdisciplinary nature creates barriers. Clear documentation is necessary to facilitate collaboration between data scientists, ML engineers, and project managers. However, different documentation practices and requirements can hinder this process. The lack of standardized methodologies impedes effective communication and collaboration between stakeholders. Moreover, the question of responsibility for documentation is often unclear, leading to scenarios where documentation tasks are neglected altogether. Ensuring that documentation is usable and comprehensible for all stakeholders, especially

those with varying levels of technical expertise, adds to the challenge. Therefore, balancing stakeholder demands while ensuring documentation is neither too cumbersome nor too superficial is a significant challenge.

Furthermore, ethical and accountability concerns also play a role in ML documentation. Documentation must include considerations of biases in data and models, as well as ethical implications of ML applications. Comprehensive and transparent documentation practices are essential to address these concerns, but they are often lacking in current practices.

While the existing documentation tools offer significant advantages, the challenges of inconsistency, incompleteness, and difficulty in maintaining comprehensive documentation contribute to the overall lack of sufficient documentation in ML projects. Significant improvements are needed to ensure that documentation is both comprehensive and accessible. There is a critical need for ML-specific automated documentation tools that can cover the entire process and dynamically update themselves to keep pace with the rapidly evolving nature of ML systems.

4. CONCLUSION & FUTURE WORK

This paper presents the findings of a systematic review of documentation practices for ML systems, with a particular focus on the distinctive challenges and requirements associated with these systems. The integration of ML into software systems necessitates the implementation of specialized documentation approaches to ensure transparency, reproducibility, and maintainability. Effective documentation is essential for capturing the complexities of ML workflows, including data preprocessing, model development, and evaluation metrics.

This review addressed the three primary research questions. Firstly, we identified the specific documentation needs of various roles within ML projects, such as data scientists, ML engineers, and project managers. This revealed the necessity for tailored documentation strategies that cater to the unique requirements of each role. Secondly, we evaluated the current state-of-the-art solutions, including FactSheets, Model Cards, architectural diagrams, and interactive notebooks, for documenting ML systems. This provided insights into the tools and frameworks that enhance transparency and accountability in ML projects. Thirdly, we examined the main challenges in the documentation of ML systems.

Despite the advances in documentation tools and methodologies, significant obstacles remain, particularly in maintaining up-to-date and comprehensive documentation in the context of the dynamic nature of ML systems. The development of automated and standardized ML-specific documentation tools that are tailored to the diverse needs of ML teams and can adapt to ongoing data and model changes is central to overcoming these challenges. Future research should focus on creating and validating more sophisticated documentation tools that integrate seamlessly with ML workflows, enhancing usability and accessibility for all stakeholders. Additionally, consistently integrating ethical and transparency-related documentation practices across ML projects is essential. Addressing this gap by continuing to refine and innovate documentation practices will enhance collaboration, reproducibility, and efficiency in ML projects.

5. REFERENCES

[1] S. Al-Fedaghi. UML sequence diagram: an alternative model. *arXiv preprint arXiv:2105.15152*, 2021.

[2] N. Ali, S. Baker, R. O'Crowley, S. Herold, and J. Buckley. Architecture consistency: State of the practice, challenges and requirements. *Empirical Software Engineering*, 23:224–258, 2018.

[3] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software Engineering for Machine Learning: A Case Study. In *International Conference on Software Engineering: Software Engineering in Practice*, 2019.

[4] M. Arnold, R. K. Bellamy, M. Hind, S. Houde, S. Mehta, A. Mojsilović, R. Nair, K. N. Ramamurthy, A. Olteanu, D. Piorkowski, et al. FactSheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM Journal of Research and Development*, 63(4/5):6–1, 2019.

[5] N. Baracaldo, A. Anwar, M. Purcell, A. Rawat, M. Sinn, B. Altakrouri, D. Balta, M. Sellami, P. Kuhn, U. Schopp, et al. Towards an accountable and reproducible federated learning: A FactSheets approach. *arXiv preprint arXiv:2202.12443*, 2022.

[6] N. Bhuiyan. A framework for successful new product development. *Journal of Industrial Engineering and Management (JIEM)*, 4(4):746–770, 2011.

[7] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. In *IEEE Big Data*, 2017.

[8] J. Chang and C. Custis. Understanding Implementation Challenges in Machine Learning Documentation. In *Proceedings of the 2nd ACM Conference on Equity and Access in Algorithms, Mechanisms, and Optimization*, 2022.

[9] J. L. Correia, J. A. Pereira, R. Mello, A. Garcia, B. Fonseca, M. Ribeiro, R. Gheyi, M. Kalinowski, R. Cerqueira, and W. Tiengo. Brazilian data scientists: revealing their challenges and practices on machine learning model development. In *Proceedings of the XIX Brazilian Symposium on Software Quality*, pages 1–10, 2020.

[10] P. C. De Weerd-Nederhof, K. Visscher, J. Altena, and O. A. Fisscher. Operational effectiveness and strategic flexibility: scales for performance assessment of new product development systems. *International journal of technology management*, 44(3-4):354–372, 2008.

[11] M. Eggert. *Compliance Management in Financial Industries: A Model-based Business Process and Reporting Perspective*. Springer Science & Business Media, 2014.

[12] S. Farrell, A. Vose, O. Evans, M. Henderson, S. Cholia, F. Pérez, W. Bhimji, S. Canon, R. Thomas, and Prabhat. Interactive distributed deep learning with Jupyter notebooks. In *High Performance Computing: ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected Papers 33*, pages 678–687. Springer, 2018.

[13] T. Gebru, J. Morgenstern, B. Vecchione, J. Vaughan, H. Wallach, H. Daumé III, and K. Crawford. Datasheets for Datasets. *arXiv preprint arXiv:1803.09010*, 2018.

[14] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. D. Iii, and K. Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

[15] Y. Hashemi, M. Nayebi, and G. Antoniol. Documentation of machine learning software. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 666–667. IEEE, 2020.

[16] A. Heger, E. B. Marquis, M. Vorvoreanu, H. M. Wallach, and J. W. Vaughan. Understanding Machine Learning Practitioners' Data Documentation Perceptions, Needs, Challenges, and Desiderata. *Proceedings of the ACM on Human-Computer Interaction*, 6:1–29, 2022.

[17] Y. Jiang, C. Kästner, and S. Zhou. Elevating Jupyter Notebook Maintenance Tooling by Identifying and Extracting Notebook Structures. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 399–403. IEEE, 2022.

[18] A. Mavridou, E. Baranov, S. Bliudze, and J. Sifakis. Architecture diagrams: A graphical language for architecture style specification. *arXiv preprint arXiv:1608.03324*, 2016.

[19] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019.

[20] T. Mondal, S. Barnett, A. Lal, and J. Vedurada. Cell2Doc: ML Pipeline for Generating Documentation in Computational Notebooks. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 384–396. IEEE, 2023.

[21] N. Nahar, H. Zhang, G. Lewis, S. Zhou, and C. Kästner. A Meta-Summary of Challenges in Building Products with ML Components – Collecting Experiences from 4758+ Practitioners. In *2023 IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)*, pages 171–183, 2023.

[22] N. Nahar, S. Zhou, G. Lewis, and C. Kästner. Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process. In *Proceedings of the 44th international conference on software engineering*, pages 413–425, 2022.

[23] H. S. Nair, S. Misra, W. J. Hornbuckle IV, R. Mishra, and A. Acharya. Big data and marketing analytics in gaming: Combining empirical models and field experimentation. *Marketing Science*, 36(5):699–725, 2017.

[24] E. L. Oreamuno, R. F. Khan, A. A. Bangash, C. Stinson, and B. Adams. The state of documentation practices of third-party machine learning models and datasets. *IEEE Software*, 2024.

[25] M. Pushkarna, A. Zaldivar, and O. Kjartansson. Data cards: Purposeful and transparent dataset documentation for responsible ai. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1776–1826, 2022.

[26] L. Quaranta, F. Calefato, and F. Lanubile. Pynblint: a static analyzer for Python Jupyter notebooks. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, pages 48–49, 2022.

[27] S. M. Rashid, J. P. McCusker, P. Pinheiro, M. P. Bax, H. O. Santos, J. A. Stingone, A. K. Das, and D. L. McGuinness. The semantic data dictionary–an approach for describing and annotating data. *Data intelligence*, 2(4):443–486, 2020.

[28] J. Richards, D. Piorkowski, M. Hind, S. Houde, and A. Mojsilović. A methodology for creating AI FactSheets. *arXiv preprint arXiv:2006.13796*, 2020.

[29] A. Sayers. Tips and tricks in performing a systematic review. *British Journal of General Practice*, 58(547):136–136, 2008.

[30] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems*, 2015.

[31] Y. Shastri, R. Hoda, and R. Amor. The role of the project manager in agile software development projects. *Journal of Systems and Software*, 173:110871, 2021.

[32] C. Tavares, N. Nascimento, P. Alencar, and D. Cowan. Adaptive method for machine learning model selection in data science projects. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2682–2688. IEEE, 2022.

[33] P. Velasco-Elizondo. Software Architecture: Developing Knowledge, Skills, and Experiences. In *Latin American Women and Research Contributions to the IT Field*, pages 217–239. IGI Global, 2021.

[34] K. Vishnevskiy, O. Karasev, and D. Meissner. Integrated roadmaps for strategic management and planning. *Technological Forecasting and Social Change*, 110:153–166, 2016.

[35] A. Wadhwani and P. Jain. Machine learning model cards transparency review: Using model card toolkit. In *2020 IEEE Pune Section International Conference (PuneCon)*, pages 133–137. IEEE, 2020.

[36] J. Wanner and C. Janiesch. Big data analytics in sustainability reports: an analysis based on the perceived credibility of corporate published information. *Business Research*, 12:143–173, 2019.

[37] G. Wiggins, G. Cage, R. Smith, S. Hitefield, M. McDonnell, L. Drane, J. McGaha, M. Brim, M. Abraham, R. Archibald, et al. Best practices for documenting a scientific Python project. Technical report, Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), 2023.

[38] E. Woods, M. Erder, and P. Pureur. *Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps*. Addison-Wesley Professional, 2021.

[39] W. Wulandari and A. D. Y. Widiantoro. Design data flow diagram for supporting the user experience in applications. *Design data flow diagram for supporting the user experience in applications*, 25(2):14–20, 2017.