How Can Manual Testing Processes Be Optimized? Developer Survey, Optimization Guidelines, and Case Studies

Roman Haas*
Saarbrücken Graduate School of Computer Science, CQSE
Saarland, Munich, Germany

Daniel Elsner*
Technical University of Munich
Munich, Germany

Elmar Juergens
CQSE
Munich, Germany

Alexander Pretschner
Technical University of Munich
Munich, Germany

Sven Apel
Saarland University,
Saarland Informatics Campus
Saarland, Germany

ABSTRACT

Manual software testing is tedious and costly as it involves significant human effort. Yet, it is still widely applied in industry and will be in the foreseeable future. Although there is arguably a great need for optimization of manual testing processes, research focuses mostly on optimization techniques for automated tests. Accordingly, there is no precise understanding of the practices and processes of manual testing in industry nor about pitfalls and optimization potential that is untapped. To shed light on this issue, we conducted a survey among 38 testing professionals from 16 companies, to investigate their manual testing processes and to identify potential for optimization. We synthesize guidelines when optimization techniques from automated testing can be implemented for manual testing. By means of case studies on two industrial software projects, we show that fault detection likelihood, test feedback time and test creation efforts can be improved when following our guidelines.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Software testing, manual testing, test optimization

ACM Reference Format:
Roman Haas, Daniel Elsner, Elmar Juergens, Alexander Pretschner, and Sven Apel. 2021. How Can Manual Testing Processes Be Optimized? Developer Survey, Optimization Guidelines, and Case Studies. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3468264.3473922

*Both authors contributed equally to this research.

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473922

1 INTRODUCTION

Manual software testing is tedious, costly, and involves significant human effort. Yet, according to a recent survey, it is still widely applied in industry [1]. Despite the availability of advanced test automation techniques, previous research reports that manual software testing often complements automated testing [10, 26]. In fact, manual testing strategies can arguably detect other software faults than automated strategies [4]. Depending on the project's context, automation might be too costly [37], too complex [36], or even impossible [35], so there is no way around manual testing in the foreseeable future.

With an increasing number of test cases and execution frequency, due to shortening release cycles, long-running test suites impede the software development process [18, 38]. There has been significant research effort on optimizing automated testing, for example, on regression test optimization [6–8, 12, 14, 15, 25, 28, 31]. Still, only few research efforts attempt to transfer techniques such as regression test selection [5, 30], regression test prioritization [17, 22], or failure prediction [18] to *manual* software testing. Transfer is hindered, among other things, by missing required data (e.g., unavailability of code coverage information [17]): In contrast to automated testing, manual testing processes are not necessarily integrated with version control or continuous integration systems, test (reporting) frameworks, and build or code instrumentation tools. In fact, it is often precisely manual tests that hamper the rapid development of systems, so optimizing them is even more important [17, 18].

Research Gap. While the few existing studies on optimizing manual testing investigate the design and evaluation of specific techniques in specific contexts, it is unclear for which automated technique(s) an existing manual testing process is an eligible target: What data are available, easily producible, and can be leveraged in which ways? Consequently, to foster adoption of manual test optimization, practitioners need to understand *what* techniques are applicable and *how* to integrate them in their *existing* processes and infrastructure.

Solution. To address this gap, we qualitatively analyze the prevalence, characteristics, and problems of manual testing activities and processes by surveying 38 test practitioners from 16 companies and different project contexts. The goal is to discover and systematize characteristics of manual testing that deviate from automated testing and that hinder or enable optimization of manual testing.

We aim at deriving an actionable set of guidelines that empowers practitioners to quickly identify potential for optimization in their own context and reveal what researchers shall address. For this purpose, we investigate the transferability of optimization techniques from the literature and further derive techniques based on levers identified in our survey. We synthesize our findings as guidelines in the form of an *annotated manual software testing process model*, which highlights integration points for optimization techniques and summarizes associated prerequisites and caveats. By means of case studies on two industrial software projects from different domains we show that, using our guidelines, test feedback time and test suite maintainability can be improved.

Contributions. Our contributions are the following:

- *Developer Survey.* Evidence that manual testing is deliberately employed without the intention of full automation, underlining the need for optimizing manual testing. We provide quantitative and qualitative insights on how software is tested manually in practice.
- *Optimization Guidelines.* A set of guidelines rooted in an annotated process model and derived from our developer survey to implement 9 optimization techniques for manual testing. We explain how to leverage existing processes and highlight integration points.
- *Industrial Case Studies.* Demonstration of the guidelines' usefulness in two industrial case studies. We pinpoint levers that can reduce test feedback time and test creation efforts.

The survey results, analyses, and the optimization guidelines are publicly available in our supplemental repository[1].

2 RELATED WORK

Studies from 2011 and 2013 on the state of software testing practice report that more than 90% of survey participants test their software manually [9, 11]. While participants of these studies see room for improvement with regard to their testing strategy (e.g., through better automation), they lack the resources to implement these. We have argued that optimization of manual software testing processes requires attention, as it addresses such scenarios where manual testing is inevitable [35, 36] or deliberately employed [10, 26].

Test optimization is widespread in automated testing [6–8, 12, 14, 15, 25, 28, 31], but techniques are often not transferable to manual testing due to missing data (e.g., code coverage information) [18] or unsuitable testing processes (e.g., only black-box access during testing) [17]. Despite these difficulties, several researchers have applied techniques to optimize different aspects of manual testing [2, 5, 17, 18, 21, 22, 30]. For example, Hemmati et al. [17] studied prioritization for manual regression tests on releases of Mozilla Firefox. In general, these techniques are often tightly coupled to specific testing processes or rely on specific data whose availability depends on the context.

We aim at guidelines for developers and testers that identify where existing optimization techniques can be used in practice based on their associated prerequisites. In addition, we pinpoint the challenges that arise in manual testing guiding further research in this area. Therefore, in what follows, we thoroughly review existing work and collect prerequisites and caveats for common optimization techniques, as shown in Table 1. The optimization techniques are later consolidated with findings from our empirical study in Sec. 3.4 to provide a holistic view on manual test optimization.

Table 1: Prerequisites and caveats of existing techniques to optimize manual testing

Ref.	Prerequisites	Caveats
	1. Test Prioritization	
[17]	Textual test descriptions, test failure history	Less effective in traditional development approaches
[22]	Textual test descriptions, test failure and execution history, expert labels to prioritize tests, test–requirement links	Labels and links are hard to obtain in retrospective and, if available, maintenance requires discipline
	2. Test Selection	
[21]	Test traces, familiarity of testers with code base	Under-specification of tests leads to unstable traces
[5]	Textual test descriptions, static code analysis tool, program profiler	Accuracy of static analysis low (90%)
[30]	Test traces, adjustment of system to separate traces for parallel testing	Unsuitable in the case of large or frequent changes
	3. Test Gap Analysis	
[3]	Test traces, version control data	Up-to-date test traces are costly, data granularity is critical
	4. Test Case Reduction	
[18]	Textual test descriptions, test failure history	Test cases need to have similar textual descriptions and there must not exist flaky tests to enable reduction
	5. Refactoring	
[2]	Textual test descriptions, individual test steps, expert labels for test suites	Varying effectiveness depending on the test suite
	6. Test Quality Monitoring	
[16]	Textual test descriptions	Parameterization requires experience

Test Prioritization. Hemmati et al. [17] were the first to study regression test prioritization for manual black-box system testing on releases of Mozilla Firefox. They found that in agile development environments, historical riskiness (i.e., how often test cases have detected faults before) is an effective surrogate for prioritizing tests when compared to approaches based on text mining test-case descriptions. Lachmann et al. [22] proposed to use machine learning to learn from test execution history (i.e., failures and execution time), requirements coverage, and test case descriptions to prioritize manual system tests. Their approach is more effective than random ordering, but requires labels, reflecting how important a test case is, which are obtained from test experts.

Test Selection. Juergens et al. [21] report on an industrial case study that demonstrates challenges of applying test selection to manual system tests based on method-level test traces. They suggest to use a semi-automated process in practice, where testers could reduce the set of tests with domain knowledge. However, one caveat of this strategy is that testers need to know how to map code modifications to test cases, which, in general, is not the case. In addition, the common under-specification of manual tests leads to unstable test traces, which reduces the effectiveness of the technique. Eder et al. [5] propose an approach for regression test selection that harnesses static analyses of the tested system's source code and manual system tests written in natural language to recover trace links between the two. Their evaluation, performed on one system and four test cases, indicates that their technique outperforms random selection of test cases, but even 90% correctly linked source code methods may be insufficient in practice. However, calibrating and evaluating the approach still involves a program profiler, which

[1] https://github.com/manual-testing-study/manual-testing-esec-fse-21/

limits its transferability. In a case study on manual end-to-end testing of legacy Web applications, Nakagawa et al. [30] show that their simple test selection approach based on method-level test traces yields test effort reductions compared to manual selection. However, it is not suitable in the presence of frequent or large code changes due to the performance penalty of dynamic analysis.

Test Gap Analysis. Buchgeher et al. [3] describe a semi-automated approach for manual regression test selection. Although their selection technique reveals deficiencies in effectiveness, it provides practical benefits for test gap analysis, that is, finding modifications not covered by tests. Alongside, Buchgeher et al. state that selecting tests solely by code coverage leads to unnecessarily large sets of test cases, version control data is too coarse grained for their purposes, and keeping up-to-date coverage data for manual tests is costly.

Test Case Reduction. Hemmati et al. [18] investigate text mining-techniques coupled with failure history-based analysis for failure prediction of system-level manual acceptance tests. Their technique can be used for test case reduction, that is, for test suite maintenance by minimizing the test suite permanently, but also for selection and prioritization. Accordingly, their technique outperforms a naïve history-based model. It is the only work we found that explicitly states applicability for test case reduction, although such techniques are often overlapping with prioritization and selection [38].

Refactoring. Bernard et al. [2] aim at improving tool support for refactoring manual tests to increase test suite maintainability, (e.g., through guided test suite minimization). For this purpose, they employ various text mining and machine learning algorithms on test steps in test case descriptions and report time reductions for refactoring and for execution of the refactored test suite. To apply the technique to a test suite, testers have to supply complexity estimates of the test suite and refactoring objectives; results vary depending on these objectives and the maturity of the test suite.

Test Quality Monitoring. Hauptmann et al. [16] study how manual tests written in natural language often suffer from quality deficits leading to decreased maintenance and comprehension. Their case study results show that their language models are able to detect test smells, yet require parameterization based on experience with the maintenance of natural language tests.

In summary, we are unaware of any previous work that investigates which optimization techniques (see Table 1) are applicable in practice, given an arbitrary existing manual testing process. Moreover, empirical studies on the state-of-practice in manual testing are relatively outdated with the most recent one being from 2013 [11].

3 DEVELOPER SURVEY AND GUIDELINES

In this section, we provide details of our semi-structured online interview, following the suggestions of Jedlitschka et al. [20], and we derive a set of guidelines for optimization of manual testing processes that synthesize our findings.

3.1 Research Areas and Questions

With our interviews, we target several research questions (RQs) from three research areas (RAs): We are interested in the reasons for the implementation of manual testing processes, outline their characteristics, and derive viable optimization techniques.

RA_1: Rationale behind Manual Testing. First, we need to understand what kind of manual testing processes are implemented in practice, why practice relies on this resource-intense way of testing, and what hinders test automation.

$RQ_{1.1}$: Why is software tested manually and what technological and organizational challenges hinder test automation? To be able to identify suitable optimization potential, we need to summarize why practitioners rely on manual testing. Additionally, there might be technological and organizational reasons for why test automation—as one of the more obvious optimization approaches—is not used.

$RQ_{1.2}$: Which testing activities are carried out manually in practice? There are many different kinds of testing which can be performed manually. We survey what testing activities (e.g., exploratory and regression testing) are carried out manually to be able to tailor optimization approaches to different needs.

RA_2: Characteristics of Manual Testing. Second, we investigate characteristics of manual software testing, how much effort is actually invested into manual testing, and which optimization techniques are already applied in practice.

$RQ_{2.1}$: How much effort is invested into manual software testing? This research question aims at determining the optimization potential with respect to testing accuracy and costs.

$RQ_{2.2}$: How does manual software testing integrate with the development process? We want to shed light on the interfaces and interdependencies of manual testing with the development process to uncover related optimization potentials.

$RQ_{2.3}$: How are test cases selected for execution and how are tests assigned to testers? Test case selection is a well-known optimization technique for automated tests, and we investigate in which circumstances it can be used in practice. The assignment of tests to testers needs to be understood because this reveals optimization constraints that might not be relevant for automated tests.

$RQ_{2.4}$: What are technical and organizational characteristics of (sub-) systems that are tested manually? We would like to understand patterns that encourage or hinder manual testing.

$RQ_{2.5}$: Do flaky tests exist in manual test suites and, if so, how do testers handle them? Flaky tests are a well-known problem for automated tests [27]. If flaky tests are also an issue for manual testing, an optimization goal would be to reduce the test flakiness, possibly with techniques different from automated testing.

RA_3: Optimization Techniques in Manual Testing. Finally, we aim at summarizing optimization techniques for manual testing.

$RQ_{3.1}$: Do manual test teams aim at test automation? How much time do they plan to invest? We investigate whether test practitioners strive for automation of their test suite and how much effort is expected and invested for it.

$RQ_{3.2}$: What potential for optimization of manual software testing exists and what are its prerequisites? This is our core research question. In Section 2, we summarize existing optimization techniques and their prerequisites. With this research question, we enrich

our research-oriented perspective by collecting actively used optimization techniques reported by our participants. Following our previous discussion on the eligibility of optimization techniques, we also summarize associated prerequisites and caveats in practice.

3.2 Participants

In August 2020, we contacted 115 test engineers, testers, developers, test architects, test leads, and test managers of industry partners. $N = 38$ responded to our survey within two months. The response rate of 32.5% is relatively high, and we lead this back to our close partnership with our research partners. Most of our participants work in Germany, but there are also several participants from Canada (1), Italy (1), Romania (1), Switzerland (1), and the US (2). The participants work for organizations of different sizes, including medium-sized companies with a few dozen employees, as well as large organizations with tens of thousands employees. Their business domains include communication, network security, finance, health technology, public transportation, information technology, manufacturing, and hardware development.

3.3 Questionnaire and Conduct

We designed a questionnaire to address the above research questions. In Table 2, we list all survey questions, map them to our research questions, and mark open (📄) and closed (☑) questions. Most of our survey questions were open so that the participants could explain their context. We used SoSci Survey to host our questionnaire and provided it in English and German (the native languages of most of our participants). All questions were optional.

Table 2: Survey questions to answer the research questions

RQ	Survey question	Type
1.1	What advantages do you see in manual compared to automated software tests?	📄
1.1	What factors force you to test manually?	📄
1.1	What conditions and obstacles make test automation difficult or impossible?	📄
1.2	Which test activities are performed manually?	📄
2.1	How large is the manual test suite overall?	📄
2.1	How many testers are there in your project?	📄
2.1	How many test cycles take place per year?	📄
2.1	How many test cases are executed per cycle?	📄
2.1	How long does it take on average to run a test case?	📄
2.1	How long does it take to execute the entire test suite?	📄
2.2	Which events trigger the execution of a test case?	📄
2.2	Is a successful test execution an acceptance criterion for change requests?	☑
2.2	How do developers find out about test failures?	📄
2.2	When is a failed test case retested?	📄
2.3	Is the entire test suite executed in every test phase?	☑
2.3	If not, how are test cases selected and prioritized for a test plan?	📄
2.3	How are test cases assigned to individual testers?	📄
2.4	Which interfaces are used to test the system under test technically?	📄
2.4	Which technology-related challenges exist?	📄
2.4	How is the System Under Test organizationally tested?	📄
2.4	What organizational challenges are there?	📄
2.5	Are there flaky manual test cases?	☑
2.5	How do you deal with flaky tests?	📄
3.1	How should your testing process develop in the coming years?	📄
3.1	Are there considerations or specific plans to carry out tests more automatically?	☑
3.1	By when should the automation be completed?	📄
3.1	How much time is currently invested in the automation of manual test cases?	📄
3.2	Which steps need to be taken for each test case?	📄
3.2	Is the execution time recorded for each test case? If yes how?	📄

3.4 Results

To analyze the answers of the survey, we used an open card-sorting technique [19]. To this end, we looked iteratively for higher-order patterns in the open answers of participants for each question. Overall, we spent 25 × 2 hours (per open question 📄) = 50 hours on categorizing 633 answers.

We structure our discussion along our research areas and questions. For each research question, we present descriptive statistics of our closed survey questions (if applicable), followed by a summary of the identified categories and how often these were mentioned by participants. To enrich our discussion, we weave in quotations of responses where appropriate. We conclude this section with interpretations and insights we gained.

RA_1: Rationale behind Manual Testing In the following, we delineate why manual testing is still applied in industry and what prevents practitioners from automating tests.

$RQ_{1.1}$: Why is software tested manually and what technological and organizational challenges hinder test automation? Fig. 1.1 and 1.2 summarize the frequencies of given answers about reasons for why software is tested manually. They are grouped into advantages of manual testing and disadvantages of automated testing. We found that manual testing is deliberately employed not only because of comparatively low ramp-up costs and high flexibility, but also due to its broader scope: its exploratory character and the often associated intentional under-specification of tests. Accordingly, practitioners deem manual testing to be "closer to reality, more context-specific, and up-to-date" and more suitable when "complexity is high and requirements are blurry." Moreover, certain industries, such as the medical technology sector, prescribe manual testing.

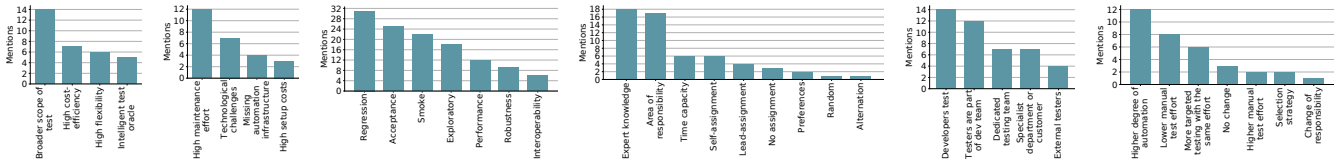
Regarding technological and organizational challenges that hinder test automation, we find the following obstacles to be prevalent (number of mentions in parentheses): Lack of time (8), lack of budget (6), limited know-how (6), limited technology (6), for example unstable testing tools or tedious creation of test data in SAP systems, interfaces to external systems (4), and high change frequency (4). One participant stated that the evolution of the software forced them to return to manual regression testing "because the [technical] environment of test automation is outdated."

$RQ_{1.2}$: Which testing activities are carried out manually in practice? Fig. 1.3 shows the frequencies of different testing activities that are carried out manually. Except for exploratory testing (e.g., including *test-as-you-code*), manual tests are specified in natural language. Manual testing seems to take place at higher levels of abstraction (integration- or system level); only two participants report conducting manual tests on the unit level.

> Summary RA_1. *According to our participants, manual tests are more flexible than automated tests in what is tested and easier to set up. They are mostly used for regression and acceptance testing.*

RA_2: Characteristics of Manual Testing Next, we explore the characteristics of manual testing processes of our participants.

$RQ_{2.1}$: How much effort is invested into manual software testing? Fig. 2 shows the distribution of manual test suite sizes and test team sizes, number of test cycles per year, number of tests per cycle, and duration of a single manual test and the entire manual test suite. The test suite sizes show a large bandwidth, between 5 and 30,000 (sic!) test cases. Interestingly, the company with the largest test suite builds software for medical devices and does not follow agile



1.1: Advantages of manual testing **1.2: Drawbacks of automated tests** **1.3: Test activities** **1.4: Assignment criteria for test cases** **1.5: Manual testing organization** **1.6: Expected evolution of the manual testing process**

Figure 1: Charts for survey answers

development practices. A test manager with a small test suite stated that "this is much too little. Since the construction as well as the recording of the test results costs a lot of time, some things are [...] tested quickly and only bugs are reported to DEV accordingly."

Also, the testing teams are of different sizes, with a median of 6 testers. The teams run from 1 to up to 40 test cycles per year, with a median of 4.5 cycles. Still, some testers indicate that these numbers may vary because "we claim to be an agile company, so it's difficult to give a number of times this process happens. " Each cycle contains, at least, 2 and up to 4,500 manual test cases, with a median of 300 test cases per cycle, sometimes this "depends on the number of change requests—for each cycle, the number of test cases differs." The median for the duration of executing a single test case is 20 minutes, and the median for running the whole test suite is 235 person hours, with a maximum of 992 man hours.

Overall, the survey responses reveal that our participants invest a lot of resources into manual testing.

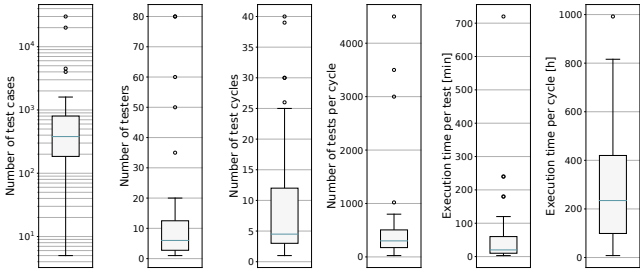


Figure 2: Distributions of manual test process characteristics

RQ$_{2.2}$: How does manual software testing integrate with the development process? Triggers for test execution are: scheduled test phases (17), finished feature tickets (16), and deployments to test environments (14). Surprisingly, more than 25% of the participants (8 out of 31 answers) state that successful test executions are not always a necessary acceptance criterion for change requests. That is, in some cases, change requests are closed even though tests failed, which might render test execution useless.

If a test has failed, 27 teams re-test directly after the code fix, while 10 teams re-test in the next test phase.

RQ$_{2.3}$: How are test cases selected for execution and how are tests assigned to testers? While 15 participants always execute the whole test suite, for example, because "from a customer point of view, we MUST run the 52 validation tests (which are appropriate to them), otherwise our software is potentially not valid for their use," some teams clean up their test suite before running it to avoid executing outdated tests, as one participant proposes: "all test cases that are not obsolete are performed in the annual test. This selection is performed every year." 20 other participants manually select particular test cases for execution. Their selection is based on code changes (6), tester experience (6), feature criticality (6), requirements (4), time constraints (4), or test failure history (3). Only 3 participants prioritize their selection explicitly, based on experience (2), or based on licensing or hazard relevance (1).

Fig. 1.4 shows how test cases are assigned to testers, where tester experience (18) and areas of responsibility (17) dominate other assignment criteria.

RQ$_{2.4}$: What are technical and organizational characteristics of (sub-) systems that are tested manually? Most of our participants run their manual tests using the system under test's GUI (28). There are also other testing environments, for example, a browser (14), hardware in the loop (HIL) (3), external systems (2), and simulators (1). Regarding tooling for running manual tests, participants adopt network communication tools (6), for example, curl, SoapUI, and Postman. Other tools mentioned by our participants are LoadRunner (1), Tosca (1), scripts (1), and Excel (2), which might also be used to manage their manual test cases.

According to our participants, the largest technology-related challenge is interference with other test environments (17), for example, because of non-isolated test systems concurrently are used by multiple testers. Frequently, there are issues because of interactions of the system under test (SUT) with other systems or applications (15), and remote test environments (12). Furthermore, different hardware combinations (4), legacy technologies (2), several test environments (2), HIL tests (1) and network latency (1) were highlighted as technology-related challenges.

Fig. 1.5 shows how manual software testing is organizationally arranged. In many cases, several groups are responsible for manual tests, for example, developers test their changes in a first stage locally on their machine before a dedicated test team verifies the changes in a later stage. Some participants report that, during a test phase, people from business departments take part in testing, still, "they all come with different enthusiasm for testing", which makes it harder for test managers to plan test activities thoroughly.

Our participants highlight many organizational challenges. One major issue is lack of time in business departments for running tests (10). Furthermore, participants point out that there is a lack of domain knowledge or testing skills (7). Additional challenges are different time zones between test and development teams (6), as well as communication and documentation challenges due to different native languages (6). For some participants, the organizational spread between test and development over different organizations (2) is another challenge. In the case of fixed release cycles, a participant complains that there is lack of time for testing (3) "because we are

the last but the release schedule is fix." That is, if anything delays the test execution, less time can be invested into solid validation. Another participant claims that—because different organizations are responsible for test and production environments—"the test environments do not match production environment enough, meaning it's possible that tests are passing but failing in production."

Other organizational challenges include different languages in specification, code and test cases (2), lack of time for training (1), coordination of testing (1), long time-to-fix (1), restricted testing environment (1), varying service providers over time (1), and, transforming development processes (1). Moreover, the domain can also pose challenges, for example, "medical technology is a strictly regulated domain", or might require special testing approaches "if I need a parallel test, there is a team session and everyone clicks on "1-2-3" at the same time." Perhaps interestingly, in the context of regulated medical technology, "agile development teams test on lower test levels", whereas manual testing is performed afterwards by the "test center for system test", implying a rather rigid (non-agile) development process such as the V-Model.

$RQ_{2.5}$: Do flaky tests exist in manual test suites and, if so, how do testers handle them? Flaky tests—tests that may non-deterministically fail and pass with the same program version—are a well-known problem for automated tests [23, 27]. They are commonly first detected if a previously passing test, that is clearly unrelated to code changes introduced to the system, suddenly fails [23, 24]. Most of our participants report that they do not encounter flaky tests (20), while others report that flaky results appear from time to time (11)—5 participants are not sure whether there are flaky tests in their test suite. Only 5 participants answered the question about how they deal with flaky tests: 2 participants re-run tests that are deemed flaky. 3 do nothing, because deviating results are "explained away", another participant puts this more diplomatically, "most of the time, the deviation turns out to be an unnoticed difference in the procedure or in the data. The tests are intentionally described vaguely in the procedure in order to cover different procedures that are supposed to produce the same result."

> Summary RA_2. *Our participants use manual tests extensively. More than half of the participants manually select only subsets of tests for execution. Tests are often assigned on the basis of experience of testers and areas of responsibility. There are many technological challenges including non-isolated test environments and the interaction of the system under test with other systems. Moreover, there are organizational challenges including lack of domain knowledge in testing teams and lack of testing skills in business departments.*

RA_3: Optimization Techniques in Manual Testing

$RQ_{3.1}$: Do manual test teams aim at test automation? How much time do they plan to invest? Finally, we report on automation and optimization potentials identified in our survey. Fig. 1.6 shows how our participants expect their manual testing process to evolve. Most frequently, a higher degree of automation is desired (12) and lower manual test efforts are expected (8). One of our participants appears to be quite frustrated about low investments into software testing, because she feels that "testing is somehow out. Nowadays, everyone tells us that a bug will simply be fixed when it appears in production." But there are also many participants who expect more targeted testing with the same effort (6) or even higher manual test efforts (2). The participants mentioned two process optimizations: implementation of a selection strategy (2) and a change of responsibility for testing (1), that is, a "shift left of our automated test cases from downstream quality assurance to development." Only a few participants (3) expect no change.

In our survey, we explicitly asked whether our participants aim at automation of their manual tests so that it becomes clear whether the implementation of additional optimization techniques can pay off in the long run. Only very few aim at automation of the entire manual test suite (2). Most participants aim at automation of some manual tests only (20). One participant points out that their goal is the "optimization of test efforts—this can mean automation, but does not have to be." Some participants also aim at no automation at all (9). Contrary to our intuition, even though most of our participants are repeatedly testing their SUT via its GUI, there are technical and organizational reasons for not automating manual tests: For instance, "frequent changes on the GUI" that disallow maintaining automated GUI tests and, according to a participant, it is "difficult to predict how much effort automation will cause because sometimes a small thing only works with extreme effort and therefore makes it difficult to plan."

For those who aim at (partial) automation of their test suite, we asked two additional questions to learn about their automation schedule and investments into test automation. 21 participants answered the question on when the automation is planned to be finished, but most of them have no specific plan when automation will be finished (18). The 3 participants who have a schedule plan to finish the automation of their test suite within the next 1–3 years.

Regarding the resources that are currently invested into test automation, only few invest, at least, one full-time equivalent (4). The other participants claim that, at least, one person works one day per week (5) or, at least, one day per month (5) on test automation. A handful of participants is investing no effort into test automation (5), even though they plan to automate tests in the future.

$RQ_{3.2}$: What potential for optimization of manual software testing exists and what are its prerequisites? In Section 2, we have approached this question from a *scientific perspective* by reviewing existing work on manual testing. This way, we have identified six techniques listed with their associated prerequisites and caveats in Table 1.

From our empirical study—taking a *practical perspective*—we identify further levers for optimization and derive respective optimization techniques: First, several participants report that there are test steps that need to be taken for each test case. Among these login to the SUT (14), creating and loading test data (10), and setting up the SUT (5) are the most common. However, only a single participant noted that they have "tests for which recurring activities are modeled with shared steps." Hence, we identify an optimization lever as the prevalence of *repeated, similar test steps*, which could be tackled by *re-using test steps*, (e.g., by means of shared test steps). This can reduce duplication and increase test suite maintainability.

Second, we found that 9 participants track the test duration either manually (2) or automatically (7). It would not make sense to track it if it did not vary among test cases and test runs. $RQ_{2.1}$

suggests that there is, in general, a large spread in test duration. At the same time, in $RQ_{2.3}$, we found that time capacity is among the three most common test case assignment criteria. Consequently, we deem the prevalence of *varying test duration* between tests to be an optimization potential that can be exploited by *test schedule optimization*: If test duration is recorded and varies, a test schedule can be generated that meets time capacity, resource, or test precedence constraints, while minimizing total testing time. Since test execution scheduling has been studied for automated testing already [29], the most straight-forward approach is to transfer these techniques to manual testing and to study their effectiveness.

Third, throughout our survey and specifically in $RQ_{1.1}$, we observed that non-exploratory manual tests are often deliberately under-specified to nudge exploratory testing. This sounds contradictory at first, because it potentially leads to non-determinism and false-negative or false-positive test results; but it seems to be one of the most popular features in manual testing, as one test case can express an entire equivalence class. Thus, one optimization lever would be to implement *flexible execution paths and test oracles* that allow the design of under-specified test cases which are still useful.

Table 3 lists the optimization levers that we identify, 3 derived optimization techniques with associated prerequisites and caveats. Together with Table 1, these make up the set of techniques that we integrate in a manual testing process model in the next section.

SUMMARY RA_3. *The overwhelming majority of our participants does not plan to automate their entire manual test suite; GUI test automation is often no option for technical and organizational reasons. Therefore, optimizing their manual testing processes is advisable. From our study, we identify 3 optimization levers.*

3.5 Guidelines for Optimization

We aim at a set of actionable guidelines that empowers practitioners to quickly identify optimization potential in their context. Therefore, we have collected characteristics of manual testing processes in our survey. In addition, we have collected and derived optimization techniques for manual testing with associated prerequisites and caveats from related work and practice (see Tables 1 and 3).

To embed these findings into an actionable set of guidelines, we devise an annotated empirical process model for manual testing. We modelled the testing processes described in the survey answers and merged them into one general manual testing process model. Although based on the empirical findings from our study, we deliberately keep the process model generic to allow practitioners to easily adopt it to their needs. We use a standard

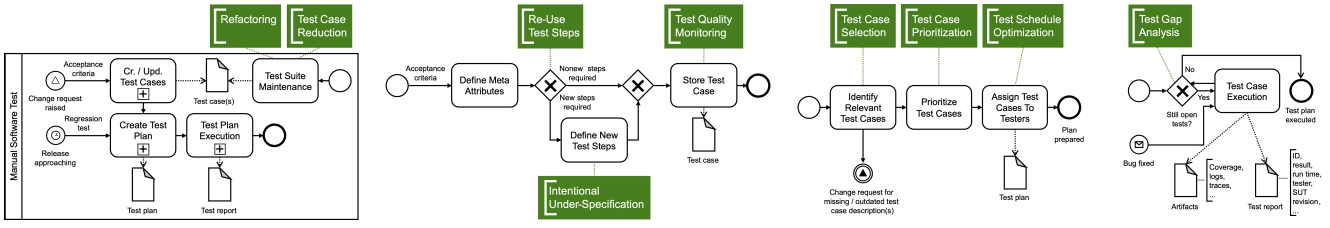
Table 3: Prerequisites and caveats of derived optimization techniques based on identified existing levers in practice (extension of Table 1)

Levers	Prerequisites	Caveats
7. Re-use Test Steps		
Repeated, similar test steps	Possibility to identify and manage test steps	Over-use of *shared* test steps
8. Test Schedule Optimization		
Varying test duration	Measuring and documenting of test duration	Time constraints, expert knowledge constraints
9. Intentional Under-specification		
Flexible execution paths and test oracles	Deterministic test oracles per execution path	False positive or negative test results, flaky tests

business process modelling notation (BPMN) to model the specifics and variety of manual testing processes that were described by our participants. Practitioners can instantiate the process model by identifying events, actions, message flows, and artifacts of their testing process. Based on this instantiation, practitioners are guided in the selection of the optimization techniques that are most relevant to them, for example, because they address current bottlenecks in their process. Using Tables 1 and 3, specific optimization approaches can be selected, based on prerequisites and acceptable caveats, and implemented in their process. For example, in the case of manual regression testing, the trigger of the manual software testing process might be an approaching release. In Fig. 3.1, this triggers the sub-process *Create Test Plan*, which is unrolled in Fig. 3.3, and its first activity is the identification of relevant test cases. The annotation shows that test case selection techniques can be used to optimize this activity. Table 1 collects prerequisites and caveats of three test selection strategies, and it guides practitioners in their assessment of the applicability of the strategies in their context.

Manual Testing Process. Depending on the specific test process, there are different start events (◯▶) that trigger the manual testing process (i.e., acceptance criteria or a scheduled regression test phase—other manual testing activities can also be covered by our model). Activities in Fig. 3.1 labelled with ⊞ are sub-processes, which are explained in more detail in the following paragraphs and figures. The optimization techniques *Refactoring* and *Test Case Reduction* can be applied most easily during test suite maintenance.

Test Case Creation. Fig. 3.2 depicts the *Test Creation* sub-process. Test steps can be *Re-Used* when tests are specified and require the same steps that are already documented for an existing test. When new test steps are defined, *Intentional Under-Specification* can be



3.1: Manual software test process 3.2: Test case creation 3.3: Test plan creation 3.4: Test plan execution

Figure 3: Optimization potentials (green) in the empirical manual testing process

applied. That is, the test can be defined generically such that several cases are covered. For example, if there are mutliple ways to trigger a functionality, the test can deliberately not specify which way to use in the test. When the test case is stored in the test management system, the *Test Quality Monitoring* can be triggered.

Test Plan Creation. Fig. 3.3 shows the *Test Plan Creation* sub-process. Initially, the set of relevant test cases that should be executed needs to be identified, optionally using *Test Case Selection*. Next, a prioritization of test cases needs to be done (which can be optimized using *Test Case Prioritization*). Finally, tests need to be assigned to testers where *Test Schedule Optimization* techniques can optimize matching testers and tests while considering relevant constraints.

Test Plan Execution. Fig. 3.4 shows the *Test Plan Execution* sub-process. The *Test Gap Analysis* can be used to determine whether test end criteria have been fulfilled. For example, it may reveal additional test opportunities from untested code changes.

4 TWO INDUSTRIAL CASE STUDIES

To demonstrate the applicability and usefulness of our guidelines, we conducted two industrial case studies with testing teams from different contexts (i.e., domain, company size, test process, and technologies). We instantiated the process models of Section 3.5 to identify applicable optimization guidelines. Together with the test leads, we then validated the suggested optimization techniques, and they decided which of these to implement. In the following, for each case study, we first introduce the study subjects to provide necessary background information. Then, we document the instantiation of the process model and, finally, we summarize the feedback of the test teams when we presented our results to them in Table 4.

4.1 Case Study 1: User Acceptance Testing

Background Information. Our first study subject is owned by Munich Re[2], an international company in the finance and insurance domain with approximately 40 thousand employees. The business information system is customized in ABAP, the custom code base counts 2.1 million source lines of code. At the time the interviews were conducted, a team of 5 testers did manual user acceptance testing (UAT). There are approximately 7 releases per year, each release has a pre-defined duration of 6–8 weeks. For each release phase, the set of change requests (*product backlog items*), which the product owner committed on and which were prioritized by the users for the current release, needs to be tested. That is, the manual software test process is triggered by new change requests, for example, by product management or users. The software life cycle management platform Azure DevOps with the plugin Azure Test Plans[3] is used to manage test cases and results.

Applicability of Optimization Techniques Following our process model in Fig. 3.1, we were able to suggest 5 optimization techniques for our first study subject, which we discuss next.

Test Case Creation. Test steps are not re-used, but structurally identical test cases are typically filed as parameterized tests for which different input and expected output values are given. This uncovers the first optimization potential: *re-use of existing test cases and steps* from former releases that have checked change requests in the same code methods. The idea is that test cases can be re-used entirely or with small modifications (e.g., new input values) if they test changes in a method that a previous test already verified. This requires that testers know which code is expected to be changed for the current change request, and testers need to be able to identify former tests that have executed this code.

The second and third optimization technique during test case creation (see Fig. 3.2) offer no additional optimization potential: intentional under-specification of tests is not applicable for user acceptance tests in this case study, as user acceptance tests are not meant to be re-executed in future release phases per se. Test cases are already automatically checked for documentation quality issues, for example, ambiguous formulations or redundancies[4].

Test Plan Creation. In the current testing process of the study subject, test cases are never re-used, which prevents optimization techniques such as test case selection, test case prioritization, and test schedule optimization. Yet, *test case selection* would help to identify relevant test cases if test cases or, at least, test steps are re-used. To benefit from *test case prioritization* and *better scheduling opportunities*, a proxy for the costs of test executions needs to be monitored, for example, the duration of test runs which is already tracked in the study subject's test management system.

Test Plan Execution. The optimization technique during test plan execution, a test gap analysis, is already used by the team[5] to reveal untested changes that should not be deployed to the production environment before a test happened.

Test Suite Maintenance. Tests are currently not re-used, so, we see no benefit of *refactoring* for this study subject. Some tests appear to be partially redundant, so *test case reduction* is promising.

Developer Feedback. Based on our recommendations, the test lead decided to implement the *re-use of test steps* in their manual testing process. Regarding the previously mentioned prerequisites, the development process has been changed as follows. First, the development team passes information on which code is planned to be changed to the test team. Second, the authors implemented test-wise coverage recording for the team, so that similar former test cases can be identified. For this purpose, the non-isolated testing environment is profiled in a user-specific way, which helps identify re-use opportunities. The testers highlighted that they like the newly created "transparency regarding which code is being executed by their manual tests." The test lead pointed out that "it would have been great to have this tool from the very beginning of the project, where even more tests were run." Now, the SUT is so large that many test runs (and thus, code changes) are needed until all actively maintained code regions are profiled. The team has started to re-use test cases where possible, even though, typically, not the entire test case can be re-used.

According to the test lead, at the end of a test phase, she again runs the *test case selection* on changes of the current release. This outputs a set of test cases that contains usage scenarios for the changed code, and thus, additional testing opportunities. Hence, she is not only using the selection as originally intended, but uses

[2] CQSE is a contracting partner of Munich Re
[3] Azure Test Plans: https://azure.microsoft.com/de-de/services/devops/test-plans/
[4] Scout: https://www.qualicen.de/scout/
[5] Teamscale: https://teamscale.com/, see also Haas et al. [13]

Table 4: Developer feedback: ✓ has been implemented, 🕒 can be implemented in the future, and × will not be implemented

Study Subject 1			
User acceptance tests	5 testers	6–8 week cycle	2.1 M SLOC ABAP
Applicable Optimization Techniques			**Feedback**
Re-use of existing test cases and steps			✓
Test case selection			✓
Test case prioritization			×
Test scheduling optimization			×
Test case reduction			🕒

Study Subject 2			
Manual regression tests	1+13 testers	12–16 week cycle	700 K SLOC C++
Applicable Optimization Techniques			**Feedback**
Test case prioritization			✓
Test case selection			×
Test case reduction			×
Refactoring			🕒
Test quality monitoring			🕒
Test plan optimization			🕒

it as inspiration for additional testing activities. She considers this helpful because it "lowers the risk of missing relevant test cases" and increases the likelihood of detecting faults before deploying the SUT to production. As far as *test case prioritization* is concerned, the test lead stated that "the order of the selected tests does not matter that much" because she "checks all selected tests to see if the team missed testing opportunities." She thinks that *test schedule optimization* "might be helpful for manual testing in general", but for her project, she doubts that "the input data is accurate enough." In contrast, she liked the idea of *test case reduction* because they often have to test similar functionality via different interfaces, and she would like to "reduce [...] redundancies."

4.2 Case Study 2: Regression Testing

Background Information. Our second study subject is an application from IVU Traffic Technologies, one of the world's leading providers of public transport software solutions. The company employs more than 700 people worldwide. We focus on the manual regression testing process for one software product (primarily written in C++, with more than 700 thousand source lines of code) that is concerned with duty planning. At the time the interviews were conducted, one tester was manually testing the product full time and 13 additional testers provided targeted testing support for releases. The test management software in use is TestLink[6], an open-source tool that is modified to suit the company's needs.

Applicability of Optimization Techniques. Again, following our process model of Section 3.5, we were able to suggest 6 optimization techniques that are applicable for the second study subject.

Test Case Creation. The test management software does not support the management of *individual* test steps, which prevents a re-use of similar test steps. Furthermore, existing tests are already deliberately under-specified to enable more exploratory testing. The first applicable optimization technique is *test quality monitoring*: Test cases of the subject are constructed using natural language descriptions, which can easily be checked by automated monitoring tools for textual quality analysis.

Test Plan Creation. Minor releases are tested only with a set of manual *smoke* regression tests (~30 test cases). For major releases, a larger test suite (~360 manual test cases) is executed, in addition. In general, there is no individual prioritization or selection of test cases. However, a subset of test cases called "developer tests", which cover substantial functionality, are first executed, to prevent blocking other test cases. As the name suggests, these tests are executed by developers during development before the testing phase begins.

Testers implicitly create a test history by marking tests as "passing" or "failing" during their execution. These test reports form a valuable artifact for optimization of the test plan. Both *test case selection* and *test case prioritization* can benefit from failure prediction models that solely rely on such information as shown by prior research on automated [8] and manual testing [17, 18, 22]. In addition, the textual descriptions could further be leveraged using natural language analyses [18, 22].

Test schedule optimization is not directly applicable, as the requirement of measuring test duration is not fulfilled, yet. However, we still assume that there are two other levers for optimized test scheduling: First, test assignments can be easily automated as they are currently manually derived from prior test plans. Second, test cases in the test management software may contain links, which define precedence or resource constraints. We propose to use existing automated techniques for generating an optimized test schedule that satisfies these constraints [29].

Test Plan Execution. Test gap analysis is infeasible, as there are no test traces recorded during testing.

Test Suite Maintenance. Since textual test descriptions and test failure history are available, the optimization techniques *refactoring* and *test case reduction* are applicable. They can be applied to create a reduced test suite that is easier to maintain, query, and extend [2].

Developer Feedback. Together with the test lead, we identified *test case prioritization* to be the most promising of the proposed optimization techniques: Accordingly, it makes sense to execute those tests first that found bugs before, "in the expectation that they will be more likely to find bugs again and thus start fixing them sooner". Therefore, we implemented a simple prioritization strategy, where tests that have failed before are executed first. This proof-of-concept already reduced the feedback time compared to the current random ordering of tests.

We decided to discard *test case selection* and *test case reduction*, as the test lead pointed out that existing test cases "in principle already represent a rather coarse-meshed coverage of the most important features", making further selection or reduction unnecessary.

Closer consideration of *test quality monitoring* and *refactoring* is generally of interest, as it is already known by the developers of the subject project that the "nature of test case descriptions has evolved over time, test cases vary widely in the quality and scope of the descriptions". However, implementing such techniques has lower priority than test prioritization inside the testing team. Finally, *test schedule optimization* is already informally done in the subject project by manually keeping track of which test cases were

[6] TestLink: http://testlink.org/

executed by which tester before. Yet, automated assistance in the test assignment could still be helpful: "It would be conceivable to provide guidance to testers in selecting unknown test cases through tags on the test cases (e.g., required specialized knowledge)."

5 DISCUSSION

5.1 Case Studies

Both test leads find the recommendations of our guidelines useful. For our first study subject, the test process and environments were changed on our recommendation derived from our optimization guidelines, so that test steps can be re-used, which saves test creation efforts. Another optimization is employing more in-depth testing because selection of former test cases is used as inspiration for additional tests. In the second case study, using our guidelines, we were able to identify and exploit the potential of reducing test feedback times by test prioritization based on test failure history.

Overall, during the case studies, our guidelines provided a goal-oriented structure for the discussion of bottlenecks in manual testing and helped the developers to focus on most relevant optimization techniques. Thus, they are well suited for discussions with testing practitioners to understand their process and tools, and helps to communicate levers of optimization techniques.

Our guidelines summarize optimization techniques that are suitable to address bottlenecks in manual testing. From our case studies, we learned that the evaluation of techniques for their practical applicability is guided well by the presented prerequisites and caveats. In both case studies, the guidelines have shown to be effective: For the first study subject, re-use of existing user acceptance tests has been improved, and a variety of tests has been increased by test case selection. In the second case study, feedback time could be reduced by prioritizing tests based on failure history.

Nevertheless, further research on optimization approaches for manual testing is necessary. Our guidelines can be extended towards this goal, and we are happy to receive merge requests in our supplementary repository.

5.2 Threats to Validity

Internal Validity. A threat to internal validity arises from the Rosenthal effect [32]: The framing of our survey questions could have influenced our participants, for example, by stating unbalanced advantages and disadvantages of manual testing. We chose the formulations of our survey neutrally, and we did several rounds of pretests with academic experts as well as testing professionals from our target group to reveal potential misleading formulations and misunderstandings. We refer the interested reader to our supplementary repository for more information and replication.

The set of guidelines presented in Section 3.1 is not meant to be complete. We focus on optimization techniques and levers that we have identified in our survey.

External Validity. We selected the participants of our survey from a small target group. We deliberately chose this group because, this way, we could validate answers and clarify open questions with participants to get a better and clearer understanding of manual testing processes in industry. Nonetheless, manual testing might be used in other ways, which limits the generalizability of our results—a common issue in empirical software engineering [34]. In particular, answers given in the survey indicate context-specific challenges, such as regulations for the development of medical technology or complexity of GUI testing, which need to be further investigated. Still, the different project backgrounds and processes provide deep insights into the variety of manual testing.

From the survey answers, we derived an empirical process model, which might not be applicable to every testing process. Yet, our two case studies show that the optimization levers and techniques, as well as their prerequisites and caveats are helpful for practitioners to identify optimization potentials in their testing processes.

In general, case study research [33] is not meant to generalize, but our two case studies nonetheless demonstrate the potential of our guidelines to assist developers and test professionals in identifying useful optimization techniques for their manual testing process.

6 CONCLUSION

Manual testing is widely used in industry, despite the high cost of the human effort required. With increasingly short software release cycles while operating large manual test suites, there is a growing need for optimization of manual test processes. Yet, existing optimization techniques from automated testing are often not directly transferable, because it is unclear how to integrate them into manual testing processes and required data are often missing. Since there is no precise understanding of the practices and processes of manual testing across industry, pitfalls and optimization potential are generally unknown.

We have surveyed 38 testing professionals from 16 companies and different project contexts to qualitatively analyze the prevalence, characteristics, and problems of manual testing activities that enable or hinder optimization. The result of this empirical study is a set of guidelines embodied in an annotated process model that implements 9 optimization techniques for manual testing. We discuss prerequisites and caveats for each technique, as they have been described in the literature or reported by practitioners during our study. We further demonstrate by means of two large-scale industrial case studies that our guidelines are useful and actionable to identify untapped optimization potential. Our two case study subjects implemented the re-use of tests, test case selection, and test case prioritization techniques. According to the test leads of our study subjects, their teams benefit from a higher likelihood of detecting faults, a reduced test feedback time, and an increased re-use of manual test cases.

As manual testing will be applied in industry in the foreseeable future without the intention of full automation, as is confirmed by our developer survey, we deem optimization of manual testing to be of significant relevance in practice. Hence, future work shall investigate the identified and proposed optimization techniques in varying settings, especially since existing studies on manual testing are still rare and not unequivocal in their results.

ACKNOWLEDGMENTS

This work was partially funded by the German Federal Ministry of Education and Research (BMBF), grant "SOFIE, 01IS18012A", and the German Research Foundation (DFG), grant "AP 206/14-1". The responsibility for this article lies with the authors.

REFERENCES

[1] Y. Baron and K. Yitmen. 2018. *ISTQB® Worldwide Software Testing Practices*. ISTQB. Retrieved July 8, 2021 from https://www.istqb.org/documents/ISTQB2017-18_Revised.pdf

[2] E. Bernard, J. Botella, F. Ambert, B. Legeard, and M. Utting. 2020. Tool Support for Refactoring Manual Tests. In *Proceedings of the International Conference on Software Testing, Verification and Validation*. IEEE, 332–342. https://doi.org/10.1109/icst46399.2020.00041

[3] G. Buchgeher, C. Ernstbrunner, R. Ramler, and M. Lusser. 2013. Towards Tool-Support for Test Case Selection in Manual Regression Testing. In *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 74–79. https://doi.org/10.1109/ICSTW.2013.16

[4] I. Ciupa, B. Meyer, M. Oriol, and A. Pretschner. 2008. Finding Faults: Manual Testing vs. Random+ Testing vs. User Reports. In *Proceedings of the International Symposium on Software Reliability Engineering*. IEEE, 157–166. https://doi.org/10.1109/ISSRE.2008.18

[5] S. Eder, B. Hauptmann, M. Junker, R. Vaas, and K.-H. Prommer. 2014. Selecting Manual Regression Test Cases Automatically using Trace Link Recovery and Change Coverage. In *Proceedings of the International Workshop on Automation of Software Test*. ACM, 29–35. https://doi.org/10.1145/2593501.2593506

[6] S. Elbaum, A. Malishevsky, and G. Rothermel. 2000. Prioritizing Test Cases for Regression Testing. In *Proceedings of the International Symposium on Software Testing and Analysis*. IEEE, 101–112. https://doi.org/10.1145/347324.348910

[7] S. Elbaum, A. Malishevsky, and G. Rothermel. 2001. Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization. In *Proceedings of the International Conference on Software Engineering*. IEEE, 329–338. https://doi.org/10.1109/icse.2001.919106

[8] S. Elbaum, G. Rothermel, and J. Penix. 2014. Techniques for Improving Regression Testing in Continuous Integration Development Environments. In *Proceedings of the International Symposium on the Foundations of Software Engineering*. ACM, 235–245. https://doi.org/10.1145/2635868.2635910

[9] E. Engström, P. Runeson, and A. Ljung. 2011. Improving Regression Testing Transparency and Efficiency with History-Based Prioritization—An Industrial Case Study. In *Proceedings of the International Conference on Software Testing, Verification, and Validation*. IEEE, 367–376. https://doi.org/10.1109/ICST.2011.27

[10] E. Engström, P. Runeson, and M. Skoglund. 2010. A Systematic Review on Regression Test Selection Techniques. *Information and Software Technology* 52, 1 (2010), 14–30. https://doi.org/10.1016/j.infsof.2009.07.001

[11] V. Garousi and J. Zhi. 2013. A Survey of Software Testing Practices in Canada. *Journal of Systems and Software* 86, 5 (2013), 1354–1376. https://doi.org/10.1016/j.jss.2012.12.051

[12] M. Gligoric, L. Eloussi, and D. Marinov. 2015. Ekstazi: Lightweight Test Selection. In *Proceedings of the International Conference on Software Engineering*. IEEE, 713–716. https://doi.org/10.1109/icse.2015.230

[13] R. Haas, R. Niedermayr, and E. Juergens. 2019. Teamscale: Tackle Technical Debt and Control the Quality of Your Software. In *International Conference on Technical Debt*. IEEE, 55–56. https://doi.org/10.1109/TechDebt.2019.00016

[14] M. Harrold, R. Gupta, and M. Soffa. 1993. A Methodology for Controlling the Size of a Test Suite. *Transactions on Software Engineering and Methodology* 2, 3 (1993), 270–285. https://doi.org/10.1145/152388.152391

[15] M. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker. 2001. Empirical Studies of a Prediction Model for Regression Test Selection. *Transactions on Software Engineering* 27, 3 (2001), 248–263. https://doi.org/10.1109/32.910860

[16] B. Hauptmann, L. Heinemann, R. Vaas, and P. Braun. 2013. Hunting for Smells in Natural Language Tests. In *Proceedings of the International Conference on Software Engineering*. IEEE, 1217–1220. https://doi.org/10.1109/ICSE.2013.6606682

[17] H. Hemmati, Z. Fang, and M. Mäntylä. 2015. Prioritizing Manual Test Cases in Traditional and Rapid Release Environments. In *Proceedings of the International Conference on Software Testing, Verification and Validation*. IEEE, 1–10. https://doi.org/10.1109/ICST.2015.7102602

[18] H. Hemmati and F. Sharifi. 2018. Investigating NLP-Based Approaches for Predicting Manual Test Case Failure. In *Proceedings of the International Conference on Software Testing, Verification and Validation*. IEEE, 309–319. https://doi.org/10.1109/ICST.2018.00038

[19] W. Hudson. 2012. Card Sorting. In *Encyclopedia of Human-Computer Interaction*. Interaction Design Foundation.

[20] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. 2008. Reporting Experiments in Software Engineering. In *Guide to Advanced Empirical Software Engineering*. Springer, 201–228. https://doi.org/10.1007/978-1-84800-044-5_8

[21] E. Juergens, B. Hummel, F. Deissenboeck, M. Feilkas, C. Schlögel, and A. Wübbeke. 2011. Regression Test Selection of Manual System Tests in Practice. In *Proceedings of the European Conference on Software Maintenance and Reengineering*. IEEE, 309–312. https://doi.org/10.1109/CSMR.2011.44

[22] R. Lachmann, M. Nieke, C. Seidl, I. Schaefer, and S. Schulze. 2016. System-Level Test Case Prioritization using Machine Learning. In *Proceedings of the International Conference on Machine Learning and Applications*. IEEE, 361–368. https://doi.org/10.1109/ICMLA.2016.0065

[23] W. Lam, P. Godefroid, S. Nath, A. Santhiar, and S. Thummalapenta. 2019. Root Causing Flaky Tests in a Large-Scale Industrial Setting. In *Proceedings of the International Symposium on Software Testing and Analysis*. ACM, 101–111. https://doi.org/10.1145/3293882.3330570

[24] W. Lam, K. Muslu, H. Sajnani, and S. Thummalapenta. 2020. A Study on the Lifecycle of Flaky Tests. In *Proceedings of the International Conference of Software Engineering*. ACM, 1471–1482. https://doi.org/10.1145/3377811.3381749

[25] O. Legunsen, F. Hariri, A. Shi, Y. Lu, L. Zhang, and D. Marinov. 2016. An Extensive Study of Static Regression Test Selection in Modern Software Evolution. In *Proceedings of the International Symposium on Foundations of Software Engineering*. ACM, 583–594. https://doi.org/10.1145/2950290.2950361

[26] A. Leitner, H. Ciupa, B. Meyer, and M. Howard. 2007. Reconciling Manual and Automated Testing: The AutoTest Experience. In *Proceedings of the Annual Hawaii International Conference on System Sciences*. IEEE, 261a–261a. https://doi.org/10.1109/HICSS.2007.462

[27] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov. 2014. An Empirical Analysis of Flaky Tests. In *Proceedings of the Symposium on the Foundations of Software Engineering*. ACM, 643–653. https://doi.org/10.1145/2635868.2635920

[28] M. Machalica, A. Samylkin, M. Porth, and S. Chandra. 2019. Predictive Test Selection. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 91–100. https://doi.org/10.1109/ICSE-SEIP.2019.00018

[29] M. Mossige, A. Gotlieb, H. Spieker, H. Meling, and M. Carlsson. 2017. Time-Aware Test Case Execution Scheduling for Cyber-Physical Systems. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*. Springer, 387–404. https://doi.org/10.1007/978-3-319-66158-2_25

[30] T. Nakagawa, K. Munakata, and K. Yamamoto. 2019. Applying Modified Code Entity-Based Regression Test Selection for Manual End-To-End Testing of Commercial Web Applications. In *Proceedings of the International Symposium on Software Reliability Engineering Workshops*. IEEE, 1–6. https://doi.org/10.1109/ISSREW.2019.00033

[31] A. Philip, R. Bhagwan, R. Kumar, C. Maddila, and N. Nagppan. 2019. FastLane: Test Minimization for Rapidly Deployed Large-Scale Online Services. In *Proceedings of the International Conference on Software Engineering*. IEEE, 408–418. https://doi.org/10.1109/icse.2019.00054

[32] R. Rosenthal and K. Fode. 1963. The Effect of Experimenter Bias on the Performance of the Albino Rat. *Behavioral Science* 8, 3 (1963), 183–189. https://doi.org/10.1002/bs.3830080302

[33] P. Runeson, M. Host, A. Rainer, and B. Regnell. 2012. *Case Study Research in Software Engineering: Guidelines and Examples* (1st ed.). Wiley. https://doi.org/10.5555/2361717

[34] J. Siegmund, N. Siegmund, and S. Apel. 2015. Views on Internal and External Validity in Empirical Software Engineering. In *Proceedings of the International Conference on Software Engineering*. IEEE, 9–19. https://doi.org/10.5555/2818754.2818759

[35] O. Taipale, J. Kasurinen, K. Karhu, and K. Smolander. 2011. Trade-Off Between Automated and Manual Software Testing. *International Journal of Systems Assurance Engineering and Management* 2, 2 (2011), 114–125. https://doi.org/10.1007/s13198-011-0065-6

[36] S. van der Burg and E. Dolstra. 2010. Automating System Tests using Declarative Virtual Machines. In *Proceedings of the International Symposium on Software Reliability Engineering*. IEEE, 181–190. https://doi.org/10.1109/ISSRE.2010.34

[37] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist. 2017. Impediments for Software Test Automation: A Systematic Literature Review. *Software Testing Verification and Reliability* 27, 8 (2017), e1639. https://doi.org/10.1002/stvr.1639

[38] S. Yoo and M. Harman. 2012. Regression Testing Minimization, Selection and Prioritization: A Survey. *Software Testing Verification and Reliability* 22, 2 (2012), 67–120. https://doi.org/10.1002/stv.430