Verdict Machinery: On the Need to Automatically Make Sense of Test Results

Mikael Fagerström, Emre Emir Ismail, Grischa Liebel
Chalmers | University of Gothenburg, Sweden
mikael.fagerstroem@gmail.com
emre.emir.ismail@ericsson.com
grischa@chalmers.se

Rohit Guliani, Fredrik Larsson, Karin Nordling
Ericsson AB, Sweden
rohit.guliani@ericsson.com
fredrik.b.larsson@ericsson.com
karin.nordling@ericsson.com

Eric Knauss, Patrizio Pelliccione
Chalmers | University of Gothenburg, Sweden
eric.knauss@gu.se
patrizio.pelliccione@gu.se

ABSTRACT

Along with technological developments and increasing competition there is a major incentive for companies to produce and market high quality products before their competitors. In order to conquer a bigger portion of the market share, companies have to ensure the quality of the product in a shorter time frame. To accomplish this task companies try to automate their test processes as much as possible. It is critical to investigate and understand the problems that occur during different stages of test automation processes. In this paper we report on a case study on automatic analysis of non-functional test results. We discuss challenges in the face of continuous integration and deployment and provide improvement suggestions based on interviews at a large company in Sweden. The key contributions of this work are filling the knowledge gap in research about performance regression test analysis automation and providing warning signs and a road map for the industry.

CCS Concepts

•**Software and its engineering** → **Software testing and debugging;** *Agile software development; Collaboration in software development;*

Keywords

Non-Functional Testing Oracle; Verdict System; Performance regression test analysis; Automation

1. INTRODUCTION

Testing activities take up between 30 and 60 percent of all software life-cycle costs, depending on critically and complexity of the product [17]. Therefore, it is important to control and reduce test costs through test automation [17]. Specifically, practices such as Continuous Integration (CI) promise to improve release frequency and predictability, to increase developer productivity and communication [18], but require test automation [13, 7] and optimization, e.g. by automatically prioritizing and selecting (most often functional) regression tests [10, 14]. Also, in a continuous integration setting, which most of the organizations either adopted or trying to adopt, testing should be run continuously [6].

As a part of software testing, performance testing aims to assess how the system shall perform, usually from a user perspective [19]. In many cases, performance degradation and difficulties in handling system throughput sufficiently well are major problem sources, as opposed to system crashes [19]. Often, these software systems have gone through extensive functionality testing but lack sufficient performance testing [19]. Despite this importance, there has not been any significant advance in performance testing and tool support remains limited [5].

To make sure that performance remains acceptable when changes are made, performance regression testing is required in addition to conventional functional regression testing [11]. Performance regression test runs usually take hours to days and create hundreds of performance indicators for a given system [11], which makes manual detection of performance regression inefficient and error-prone. In order to efficiently perform regression testing, it is required to automatically obtain a test verdict from these performance indicators. However, only few studies address this topic, causing a gap in research in the area of automated performance regression testing [11].

We address this gap by presenting the results of a case study on automated performance regression test analysis at Ericsson AB. The goals of the study are (a) to investigate challenges which occur during the automation of performance regression test analysis and (b) to suggest possible improvements for these challenges. We collected data from multiple sources, namely twelve semi-structured interviews, an analysis of an existing verdict machinery, workshops at the company, and from documents describing the system context. Our results show a number of challenges:

- Deriving a verdict from non-functional regression testing can significantly lengthen feedback-cycle time in continuous integration.
- An automatic verdict machinery promises quicker and better feedback, but needs to address the challenge of

continuously evolving features, which in turn impact non-functional properties of the system.

- Insufficient requirements as well as branching of development streams are major challenges for implementing a verdict machinery system. Thus, it becomes difficult to offer a root cause analysis and unacceptable amounts of wrong verdicts can occur.

- Acceptance of the verdict machinery was further challenged by unclear and varying expectations towards the verdict machinery throughout the case company as well as by missing standardization and central use of conventions around testing.

Yet, all interviewees agreed that automated verdicts are helpful and that the existing system should be further developed. Improvement suggestions include:

- Standardization and centralization, a testing map to give an overview of the state of affairs, and facilitation of common understanding through the tool.

- Further, the tool should be improved in terms of usability, accuracy, and by considering historical data.

The rest of this paper is structured as follows: We discuss verdicts and their relation to test oracles in Section 2 and describe the case company in Section 3. We present the concept of the automatic verdict system in Section 4, where we also give an example. We then present our research method in Section 5 and our results in Section 6. Section 7 offers a discussion of implications of our work as well as threats to validity and Section 8 concludes the paper with an outlook on future research.

2. TEST ORACLES AND VERDICTS

To automate the analysis of test runs, test oracles are required to assess whether a test has passed or not. A test oracle checks whether the result of executing a test is as expected. We distinguish four different kinds of test oracles, based on Barr et al. [1]. Formal specification languages can be used to define test oracles, which then judge the behavior of the System Under Test (SUT) according to this specification. Derived test oracles use different artifacts, such as documentation or system executions, to judge a system's behavior, when specified test oracles are unavailable. Implicit test oracles rely on general knowledge which is true in most cases, such as "segmentation faults are nearly always errors" to assess a SUT's behavior.

For instance, the work in [2] makes use of an oracle in the domain of SOAP-based web-services and the oracle builds on the following observations: (i) whenever invoking different operations with wrong input data, the error answer message is (almost) always the same; (ii) error answers are typically short; (iii) regular answers are typically different from each other; (iv) regular answers are typically long.

Finally, the responsibility to evaluate the behavior of the SUT can be placed on human testers. In many cases, this final strategy is used as it is difficult to differentiate desired behavior from potentially incorrect behavior, due to tacit knowledge, such as informal specifications, norms, or expectations.

When a test oracle observes system behavior, it returns a test verdict, which is either *pass*, *fail*, or *inconclusive*. The verdict pass indicates that the observed behavior matches the specification and the test objective has been reached. The verdict fail means that the observed behavior is a failure and is not consistent with the specification. The verdict inconclusive is used when no failures have been observed, but the test objective has not been reached. While classical test oracles observe the result of a single test run, Hierons [12] argues that a new kind of test verdict is required, taking into account multiple test runs. In our study, the case company uses such a verdict system, considering the history of multiple performance regression tests.

3. CASE COMPANY

The case company is Ericsson, a Swedish-based multinational organization offering services, software, and infrastructure in information and communication technology for telecom operators and other industries. Their portfolio includes traditional telecommunications, Internet Protocol networking equipment, mobile and fixed broadband, operations and business support solutions, cable TV and IPTV, as well as video systems. In order to meet customer expectations, the case company aims to achieve highest possible performance with respect to scalability, capacity, and similar (non-functional) criteria. According to the annual report for 2014, the company's current position is number one in mobile infrastructure, OSS and BSS, telecom services and TV platforms. The company has also the number one position in the LTE market share in the world's 100 largest cities. Other company facts include 37,000 patents, 180 countries with customers and 118,000 employees.

At the company, we studied an existing telecommunications system developed in the Evolved Packet Gateway (EPG) department (to which we also refer as the case department). EPG is a component of Evolved Packet Core, which provides a solid foundation for delivering mobile data services [8], and it is developed to be a critical part of LTE networks. EPG as a product can be valuable for service providers, as they can use it as a gateway between their mobile packet core network and the Internet [8].

The number of smartphones and smart mobile devices is rapidly increasing globally and it is expected the be over 3.7 billion by 2017 [9]. These devices enabled users to connect to Internet everywhere within a network connection and this recent phenomenon resulted with customers demanding high quality broadband. Consequently, the system under investigation has large demands on scalability, performance, and availability to service providers. In order to satisfy these demands, each new build undergoes extensive testing on an daily, weekly, and monthly basis, and recently also hourly test executions on a simulated system. In particular, the company uses an automated verdict system for test analysis. To prevent faults from slipping through to customers, potential inadequacies which are detected during testing have to be reported within the company. This makes automated testing particularly important. It is also important to note that there are on average 15 commits per hour and the performance tests are executed hourly, daily, weekly and monthly.

4. AUTOMATIC VERDICT SYSTEM

The verdict system used at the case company is a computer script that automatically analyzes test results. It can

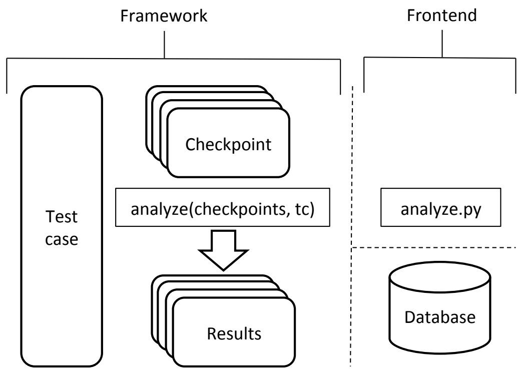


Figure 1: Architecture of the verdict system.

analyze a large number of results within seconds and its purpose is to accelerate test analysis in order to achieve a larger test coverage with the same amount of testers. The system is written in the Python programming language and has a simple loosely coupled design, which makes it easy to extend the automatic analysis. The system's architecture is shown in Figure 1. The system takes a log file of the test results as the input, and according to expected requirements, creates a verdict that can reduce into seconds a task that once done manually can take hours.

Checkpoints are interfaces which define method signatures for specific test definitions. Various checkpoints exist for different quality attributes, including capacity, robustness and stability.

Given a set of checkpoints, the verdict system analyzes a test case and, for each checkpoint, returns a test result. If all checkpoints have passed, the corresponding test case is marked as passed. A test case contains methods to read log files generated by the test execution. These contain test results in the form of Key Performance Indicators (KPIs), also called performance counters. There are hundreds of different KPIs that measure different properties of the SUT, for example CPU and memory utilization as well as throughput.

There are different kinds of checkpoints, depending on how the verdict is decided. Some checkpoints compare their KPIs with a corresponding threshold value, others compare their KPIs with KPIs from previous test executions. The former type determines whether the properties of the SUT fulfill the specified non-functional requirements, whereas the latter type aims to search for performance degradations. Checkpoints are created by testers and contain information about which KPIs within the test cases have to be analyzed (prerequisites) and what values are required to get a pass verdict.

A typical testing process is as follows. First, the analyze function takes the test case object and a list of checkpoints as an input. The function then decides if checkpoints are valid and performing up to standards based on the test case attributes. Finally, it prints the results on the screen and stores them in the database.

Consider the following example on how a capacity test case is handled through the verdict system.

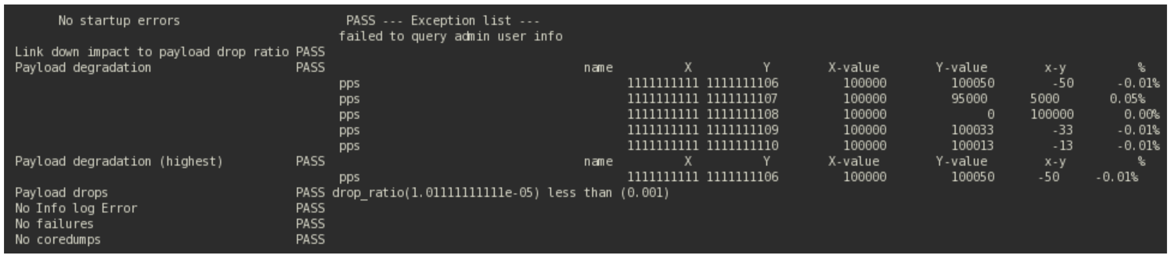
There are three main categories for capacity test cases; payload, PDP (Packet Data Protocol) and signaling. A payload test case will be the basis of this example. Even though these test runs are usually done automatically it is also possible to configure and run them manually. Figure 2 shows the result of a final verdict. The first column shows checkpoint names. Each of these checkpoints has certain requirements to be met, which the verdict system compares to the test results. The results of this comparison are listed in the second column, as PASS or FAIL in most cases, or as WARNING or N/A if there is limited information about the results or if they are not below the requirements but at a certain level. In the remaining columns, the test results in terms of numbers and comparisons with previous test executions are listed.

As the first step, the product which the test is going to be run on is selected. The product in this case is the system under investigation that provides mobile phone and internet connectivity between multiple users. Secondly, a capacity payload test case is chosen. Payload test cases usually include requirements about maximum rate of successful message delivery, i.e., throughput. The test cases hold information about what needs to be tested, how it has to be tested, and in which environment it has to be tested, preventing test execution on incompatible products. After correctly choosing the test case and the product, the test is executed. The test run produces a large quantity of data for testers to investigate, which can create a bottleneck within the testing process. This is where the verdict system helps the case company by providing a summary of all the information gathered during the test run. The system takes the requirements stated in the test case definitions as an input and scans through all available log files to gather the information necessary to evaluate the requirement. In a typical payload test case, requirements specify the minimum level of throughput, the maximum allowed degradation in throughput, and that the system is not allowed to produce unexpected error messages. If these requirements are met, the system produces a pass verdict. In case one requirement is not met, the system produces a fail verdict, even if other requirements in the same test are met.

For certain checkpoints, the verdict system goes through the SUT and tools logs for any error messages and KPI values (basically the verdict system looks within the logs for keywords such as "ERRORS" and "Value"). Once these pieces of data are compared to requirements and historical data, the verdict system makes a verdict and prints out the result as a log file. It is then the tester's responsibility to judge the verdict outcome and, if verdict shows fail, to take necessary steps to report the faults.

Interview no.	Role	Experience (years)
1	Function Tester	10+
1	Systems developer	10+
1	Software designer	10+
2	Systems manager	5
3	Systems Tester	7
3	Software developer	10+
3	Verification engineer	7
4	Operational Product Owner	10+
5	System test specialist	10+
6	Line manager	10+

Table 1: Interviewees at studied department



```
      No startup errors                    PASS --- Exception list ---
                                          failed to query admin user info
Link down impact to payload drop ratio PASS
Payload degradation                    PASS                                    name         X          Y        X-value        Y-value       x-y          %
                                          pps                                        1111111111 1111111106      100000         100050        -50      -0.01%
                                          pps                                        1111111111 1111111107      100000          95000      5000       0.05%
                                          pps                                        1111111111 1111111108      100000              0    100000       0.00%
                                          pps                                        1111111111 1111111109      100000         100033        -33      -0.01%
                                          pps                                        1111111111 1111111110      100000         100013        -13      -0.01%
Payload degradation (highest)          PASS                                    name         X          Y        X-value        Y-value       x-y          %
                                          pps                                        1111111111 1111111106      100000         100050       -50     -0.01%
Payload drops                          PASS drop_ratio(1.01111111111e-05) less than (0.001)
No Info log Error                      PASS
No failures                            PASS
No coredumps                           PASS
```

Figure 2: Log file of a successful final verdict

5. RESEARCH METHOD

To achieve the goals of this study, as stated in the introduction, we investigate the following research questions:

- **RQ1**: What are the major challenges of automatically analyzing non-functional regression test results?
- **RQ2**: How can the automation of non-functional regression test analysis be improved?

In order to answer these questions appropriately with respect to their real-life setting, we used an exploratory case study design.

5.1 Data Collection

We collected qualitative data from a combination of multiple sources, namely document studies, interviews, workshops, and an analysis of the existing system at the case company.

As a first step to familiarize ourselves with the studied department and the system under investigation, available online documentation at the case company was studied. This served as a means to avoid misunderstandings later on during data collection. In order to understand the existing verdict system and to identify suitable interviewees, several meetings were additionally held together with test architects at the case company.

We then conducted twelve semi-structured interviews[1]. Out of these, nine were individual interviews and three were group interviews. In total, we interviewed 18 individuals. To achieve variety of sources and increase validity, we chose interviewees with different roles and experience levels. Additionally, for data triangulation, six of the interviews were conducted with interviewees outside of the studied department. The range of interviewees is depicted in Table 1 for the studied department and in Table 2 for other departments or companies.

By using three different interviewee sources, i.e., interviewees at the studied department, outside of the department but at the same company, and at other companies, we were able to obtain diverse data and to study the problem from different angles. Interviewees at the case department were most familiar with the existing verdict system and its problems. Additionally, expectations, possible improvements, and plans for future improvements could also be gathered from this group.

Interviewees from the case company, but outside of the case department, provided us with company-specific information, but at the same time with a different perspective.

Employer	Interview no.	Role	Experience (years)
Case Comp.	7	Line manager	10+
Case Comp.	7	Developer	10+
Case Comp.	7	Developer	8
Case Comp.	8	I&V discipline driver	10+
Other Comp. 1	9	Quality assurance engineer	9
Other Comp. 2	10	Chief leader for continuous integration and test governance	10+
Other Comp. 3	11	Technical expert	7
Other Comp. 4	12	System engineer	10+

Table 2: Other interviewees

Finally, interviewees from other companies provided a point-of-view from outside the case company. In particular, we wanted to know whether the same or similar problems exist outside of the case company.

Interviews lasted between 45 and 60 minutes and were recorded. Additionally, notes were taken and pictures of sketches were made, where applicable.

To study the actual system and triangulate interview data, we had access to the running verdict system and its source code. In order to understand the difficulties users can face while using the system, we tested the system with different test scenarios and input data.

As a last data collection method, we attended several workshops held by the case company. The workshops took two to four hours, covering different aspects of the case department and the domain, e.g., technical details of the application domain. This enabled us to be a part of the group and interact with more experienced members.

We have used workshops and other data sources besides interviews as a way to understand how the verdict system works, how the testers interact with it in practice as well as what its potential and purpose are.

5.2 Data Analysis

All the interviews were transcribed and pictures taken

[1]Interview guide available at https://dl.dropboxusercontent.com/u/13255493/Fagerstrom-Ismail_Verdict-Machinery_interview-guide.pdf

during the interviews were mapped to the respective interview part. Afterwards, we performed coding, i.e., categorization of data into conceptual categories [4]. In order to reduce the risk of inaccuracy, multiple researchers took part in the coding process. After coding, the conceptual categories were grouped iteratively.

Analysis for the data obtained from document study, system analysis, and workshops was performed using qualitative content analysis, where data is grouped with the help of categories that are generated inductively [16]. Categorization began as soon as new information was retrieved through the different collection methods. Common keywords were collected and categorized whenever they related to the research questions. These were then grouped based on similarity. The groups were iteratively refined when new data were collected.

6. RESULTS

In the following, we present the major challenges (RQ1) and improvement suggestions (RQ2) of automatically analyzing non-functional regression tests. These findings originate from the analyzed interview data and are triangulated with additional data collected from workshops and the running verdict system and its source code.

6.1 RQ1 - Challenges regarding automation of performance regression test analysis

The seven challenges presented in this section are listed in Fig. 3. They can be sorted into technical challenges, addressing primarily technical aspects such as the possibility to support source code branching, and social challenges, addressing social aspects such as the expectations towards the verdict system.

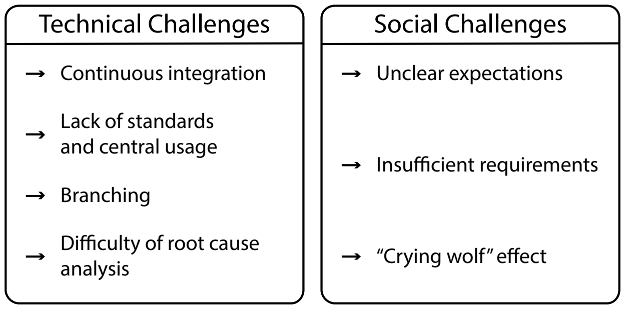


Figure 3: Thematic map of challenges.

6.1.1 *Continuous Integration*

Continuous integration emphasizes fast feedback cycles in software development, which in turn can increase the software quality significantly. However, the combined time of executing the tests and analyzing the test results in order to derive a verdict can slow down the feedback in continuous integration considerably. In this way, performance regression test analysis in the scope of continuous integration presents new challenges. While we have to consider that introducing new features can cause a decrease of the system speed and increase of the CPU usage, it is difficult to estimate the correct tolerance.

Two of the participants stated that continuous integration made it difficult to pinpoint where a fault is introduced to the system. For example, a systems manager stated:

> *"It's harder basically, if there's a capacity degradation it is hard to see where that capacity problem is introduced."* — Systems Manager

Another interview subject said that:

> *"The problem is that this daily scope is not very well tuned into continuous integration ways of working. In the continuous integration view it says that you should not do your troubleshooting when it is delivered, you should do your troubleshooting before or after it is backed out from the delivery. So, in that sense all the test cases are to just make sure that you have a working build and you can push in as much testing as possible. This is what we want to guarantee with overworking build. And if it fails, then we do not really care to look at why it fails, we merely want to know which change caused this build to fail and then we back out that delivery and we go back to the state we had before of working build and then we do the troubleshooting outside that. But the daily scope is not very well tuned to that, because in one day we can have like a hundred commits to the master branch. You might have two, three or four new failures like new core dumps, decreased capabilities, some kind of new alarms or whatever. It is very hard to find the faulty commit in that consequence."* — Operational Product Owner

Due to continuous integration and long run times of the system tests, it is possible to have tens of or at some cases hundreds of commits between two test runs. In case of a performance degradation which results in failed test cases, it is not clear which commit or group of commits actually caused the problem. Therefore, it becomes the testers' responsibility to dig into each commit and try to figure out the faulty one that is causing degradation. It is impossible to run system tests after each commit and, as stated by [6], it might be catastrophic to even try it. Moreover, the complexity of the system implies extensive analysis time for test cases, following-up errors, taking logs and writing problem reports. This causes problems in the continuous integration way of working. Automation of this analysis (e.g. by applying the automatic verdict system) promises to accelerate feedback in continuous integration, but has to address these challenges.

6.1.2 *Lack of Standards and Central Usage*

We observed some naming inconsistencies within the tool that is used at the case company. Moreover, as can be seen in the following note, two participants pointed out that they had problems with naming, lack of standards, and understandability about the checkpoint content:

> *"Since on the top-level you only get this very cryptic checkpoint name, so it is just letters and numbers."* — Operational Product Owner

The use of code names and abbreviations made it difficult for testers to evaluate the problem initially and required them to examine other available material such as logs to understand the system's output. A line manager complained about the lack of a central usage about the tool:

> *"It would be better to have a centralized place for all the non-functional requirements in the verdict system, instead of having multiple of the requirements outspread in different checkpoints. This improvement will increase the performance, usability, and maintainability of the system. In order to do this improvement the architecture of the system has to be changed."* — Line Manager

6.1.3 *Branching*

Branching constitutes the underlying reason for a big portion of the problems in the analysis process. During the development process, developers branch out on one of the previous builds to work on their local repositories. After the coding is over, developers commit their changes back again to the central repository where performance regression tests are done. However, this way of working is prone to several issues as requirements, boundary values, and acceptance levels for each build can be different from each other.

On the one hand, a developer who branched out on a certain build is essentially affected by the performance of that particular build. Even though the changes the developer made within that code is up to standards and optimizes the performance, due to initial low quality of the code, the final version might not perform up to standards according to performance acceptance tests and thus, be labeled as failed.

On the other hand, the other way around might also be problematic. If a certain developer commits a piece of code that affects the performance negatively, it might go unnoticed if the build that was branched out on is an exceptionally performing one. In this case, the performance of the final build will be up to standards, but it could have been even better.

It was also noted that this issue is related to the continuous integration way of working and insufficient requirements. Several developers working with a number of different branches make it difficult to automate the test analysis process, since it will be a major challenge to get the updated requirements and acceptable boundary values for performance indicators.

6.1.4 *Difficulty of Root Cause Analysis*

Several interviewees mentioned multiple times the difficulty of finding the root cause of an error or a failed test case during test analysis. Due to several reasons, such as continuous integration, complexity of the SUT, high volume of available data, and the large number of developers working with different branches, it is difficult to locate where the problem is first introduced to the system. A system developer stated that the verdict system only indicates the existence of something wrong in the system, not what it is. It is the developer's responsibility to understand the cause of that problem. Some participants also stated that such attempts might be futile. As the verdict system was not designed initially to locate errors, it would take a huge amount of effort and time to integrate that functionality. One product owner stated that:

> *"Well, if you spend 1000 hours just on trying to implement a program that find the root cause of this specific problem, then you have another problem next week."* — Operational Product Owner

6.1.5 *Unclear Expectations*

Our data shows that participants had different goals and expectations from the performance regression test analysis and its automation process. Even though some interviewees agreed that only limited functionality can be expected from the used tool, others demanded more functionality and capabilities. Three interviewees stated their desire to use the tool as a warning system for possible problems that do not hold any risk in the short term, but could cause issues in the long term. They stated that the indicators were there for a manual tester to see and it was possible for an automated system to pick up these signs and warn testers about the decreasing quality of the SUT. In that case the SUT would perform up to the required standards, but since it performed worse with every new build, the tool would warn the testers about the possible risks. All of these different expectations point out that what is demanded from this process can lead to disagreements and tools that are devised to solve this problem can not satisfy all of the stakeholders. As stated by [15], it is difficult to align goals within large organizations, such as the case company, as each different department has their own perspective and goals to think about. While different satisfaction levels for the tool do not affect its performance, this might affect the morale and incline to use the tools for employees in some of the departments.

6.1.6 *Insufficient Requirements*

One of the challenges for automation of performance regression test analysis was that requirements for either passing or failing a test case were not always sufficient. Due to high number of test cases, KPIs, and dynamic test environments, existing requirements could be insufficient or outdated after a while:

> *"One other thing also with system test is that not everything is specified like a black and white true or false. For example, if we restart on a card and we have a requirement that says: it should be up and running within ten seconds. However, after a software delivery it could be 11,5 seconds, so, is that totally wrong or is it OK? The increased time could depend on a number of things. It could depend on one other feature increasing the load on the system. You have to ask someone, who says: OK, we can live with it and then we change the checkpoint to say 12 seconds is the new limit."* — Operational Product Owner

The results show that missing and outdated requirements pose a great threat for the test analysis automation process. Testers either had to decide on some values based on their experience or had to contact other stakeholders to get more information. The dynamic and complex nature of the performance regression testing process forces testers to adapt to the new situation and to update the requirements and expected values for test data themselves. This finding correlates with [3], where frequent changes in requirements are mentioned as a reason to force testers to find and agree upon the correct version of requirements, thus they result in additional work and loss of time.

6.1.7 *"Crying wolf" Effect*

Our data indicates that accuracy of the verdict system is a major challenge. The tool at the case company can sometimes give false positive or false negative verdicts about a certain test case. Test architects and testers agreed that false positive cases happen more often that false negatives. Three interviewees stated that when these mistakes start to happen regularly, people responsible for investigating the failed test cases can lose motivation and concentration. A software designer stated:

> *"If you always get "Fails", you lose confidence in verdict tool, you lose motivation to carefully take a look into the problem."* — Software Designer

Several anecdotes were shared during the interviews with respect to this issue. Here, we report one example of false negative and one of false positive, respectively.

False positive: In one scenario, a developer knows that there will be a hardware upgrade next week for the SUT, which will raise the acceptable value for the memory consumption KPI. So, this developer integrates new functionality to the SUT, even though he is aware that the memory consumption would be more than what is permitted at that time. When the SUT is tested, according to initial acceptable memory consumption levels the test has to be failed and thus verdict system fails the test case. However, testers are also aware of this situation and decide that verdict for this test case has to be overruled and they pass the test.

False negative: Another case that has been discussed is also about the memory consumption. In this scenario, the developer commits the new code into the system and it has seen that memory consumption is in the acceptable levels, so verdict system decides that test case is passed. However, when manual analysis is made the testers saw that there is a big degradation after this commit and the memory consumption would be more than accepted in the future. Thus, they have decided to failed the test case even though KPIs were in the acceptable boundaries at the time. These scenarios are two examples of how the test analysis process is dynamic and requirements are prone to change.

6.2 RQ2 - Suggested automated performance regression test analysis improvements

6.2.1 *Standardization and centralization*

Three interviewees argued that having consistent standards for logs, test cases, and documentation is an important enabler for verdict systems. A technical expert suggested that:

> *"If you standardize the way of describing certain data, if you want the easier performance, standardize how to write a log file, so you can search for patterns that could be easier. Because you want to identify, let's say you have 50 different computer systems and you want to trace the CPU performance for all those and you have no clue how to find that, then you could give them the same pattern, the you can track them."* — Technical Expert

It can also be argued that by assuring standardization, testers can become more capable to locate issues and to troubleshoot. Centralization also plays an important part in the standardization process. Many interviewees complained about not being able to use some of the tools, test cases , and checkpoints because they belong to only a single person or a group. A system test specialist suggested:

> *"I think it is better to have checkpoints and they belong to everybody. So, we can just for pick any of them, except that we are not using it like that actually."* — System Test Specialist

A possible improvement is to create a centralized repository where all test cases, checkpoints, and different tools were collected and belong to everybody.

6.2.2 *Testing map*

Four interviewees stated that having a testing map could make it easier to find the root cause of the errors. A chief leader for continuous integration stated that:

> "[..] *we need to combine with resources from different test paths and link these to different releases, because a feature can be released in many different products and tested in test paths and these two need to be linked together. So that we can see a complete picture of a feature on a given release. That's one thing we need to work on also."* — Chief Leader for Continuous Integration

The lack of information about the connection between a certain test case and the branch which the SUT was built upon inhibits the testers' ability to find out how and why the test case had failed. Storing this piece of information and presenting it to the tester for each test case would improve the usability of the verdict system and makes it redundant to use other tools for the testers. It was also been stated that the information about the test cases corresponding to a certain core dump does not exist. A verification engineer suggested that presenting this piece of information could benefit the system.

6.2.3 *Common understanding*

It was stated by four interviewees that the goals and expectations for the processes and tools have to be discussed and agreed upon by all of the stakeholders. Unclear goals result in confusion for developers and testers, as expectations and responsibilities become unclear. Tools used are also criticized for not being dynamic enough and lacking functionalities that make it easily adaptable to changes. Interviewees mentioned that requirements were not dynamically updated and clearly defined in some cases and one way to solve this could be introducing rationales. Linking rationales with both the goals for the tools and requirements can help the team to be more blending to unforeseen challenges. A software developer stated that accuracy of the tools used depend on clear expectations and goals:

> *"My opinion of this verdict tools is that we need to set very clear expectations, so that this system can judge if a checkpoint is passed or fail. This is the reason why it is not very dynamic toward any change or anything else in the product can make it fail."* — Software Developer

It was mentioned that both expectations of the tools and performance of these tools can be improved with up-to-date requirements, rationales for decisions, clear goals , and expectations. Two participants also stated that understanding the needs and expectations is essential for designing an automated solution for performance regression test analysis. It is critical to understand what the customer and stakeholders expect and use that information to validate the test cases. A quality assurance engineer explained that understanding why the data is needed in the first place is also crucial:

> *"You need to understand what kind of data we are working on, in what context it should work, what is the expectations of the users."* — Quality Assurance Engineer

In order to reduce misunderstandings and set corresponding goals and expectations for the tools, it is essential to have an open line of communication between testers, developers and other stakeholders. The complex nature of the SUT and test environment at the case company forces testers to make quick and critical decisions and adapt to sudden changes. Interviewees stated that in order to manage the changing expectations and be adaptable, every tester should be comfortable with the dynamic nature of the test analysis phase. One way to achieve this is to define goals along with the rationales which are agreed upon in the beginning with all of the stakeholders. Rationales would help individual testers to make quick decisions on the spot and as the next step would provide more information to implement into the tools for the automation process. We see similarities with the findings of Bjarnason et al. [3], who state that requirement rationales can help synchronization by better supporting passing the responsibility between the different roles, in this case to the testers. Requirement rationales would help testers to be more proactive in their approach when test cases fail. Currently, testers let developers handle some of the responsibility to find the root cause when a test does not pass. This behavior puts unnecessary workload on the developers as in most of the cases the test fails due to external causes, not because the SUT was not performing up to standards. Agreeing upon initial goals and having requirement rationales might help reduce this problem, as testers would start making decisions on their own. Then, these decisions can be passed on the tools which can help automating the test analysis process.

6.2.4 Tool Improvement

Three interviewees mentioned that creating a new checkpoint or editing an existing one has to be made easier. They argued that while it is easy to use an existing checkpoint as a template, inserting some new additions such as constraints to that template makes it more difficult. Also, in order to create a new checkpoint, a tester has to read the corresponding API which makes it even more challenging. A software developer suggested that by building a new interface for checkpoint creation, time spent on this activity could be lowered and new employees can easily use and/or create checkpoints without the need for training or study sessions. The testers would be able to use this interface to create or edit checkpoints much easier than before. One of the issued raised most frequently about the verdict system was late feedback. Testers have to wait until the testing process is completed to verify a test case and it is not possible to do a real-time verification. A quality assurance engineer suggested that having a snapshot of the process can help to save time:

> *"One weakness I feel right now is that we lack the ability to verify the test while it's running. We lack real time verifying test results. If we divide in the middle, we can have take some snapshots and see if it's broken, but right now we don't have that."* — Quality Assurance Engineer

Receiving early feedback can be critical and save some time during the process. A change manager stated that having quick feedback loops could provide more information in acceptable time frame, thus allowing testers to know if the changes in the system caused any problems. Two interviewees stated that improving documentation for several different components could improve the general usability of the verdict system. Having a consistent documentation throughout the whole system would increase consistency and understandability. This certainly applies for the code documentation. A software test engineer mentioned:

> *"It can always be better documented. The code is written in Python and it has a few syntaxes that are on a very high or an expert level. It is not very easy for beginner Python programmers to understand how it works, so that is perhaps something that could be improved."* — Software Test Engineer

A technical expert suggested that test cases also need to be written in a more descriptive way. Again, this would allow new testers to grasp how certain test cases work, thus easing their integration into the testing team.

6.2.5 Statistical Intelligence

One common point of suggestion to improve accuracy was integration of a statistical intelligence component inside the verdict system. A function tester stated:

> *"In my ideal world, we should be able to produce a verdict system that sorts out this study with the help of statistics. It doesn't have to have the intelligence that we have but it is very easy to see this problem."* — Function Tester

The verdict system fails to see degradation which a human tester has no problem seeing and to detect future regression for the SUT. A software designer suggested that using historical data along with statistical analysis can help detect performance regression:

> *"In my ultimate world, you will make a verdict script that can filter out the noise and trigger the degradation that is statistically proven to be a degradation based on history."* — Software Designer

A verdict system with statistical intelligence can have more accurate verdicts and at the same time warn testers about abnormalities which can cause problems in the future. These abnormalities also include expected degradation. It was mentioned that sometimes degradation is wanted and expected, especially if new functionality has been recently delivered for a SUT. With a statistical analysis verdict system, it becomes possible to detect whether there is no regression in performance, which could also be troubling.

6.2.6 *Historical Data*

In order to improve accuracy for automated performance regression test analysis at the case company, most of the interviewees suggested that historical data has to be used more extensively within the verdict system. The verdict system at the case company only takes the last five results into account during the analysis process. A function tester stated this was not enough and more data should be used when making a decision on a test case:

> *"If there's a history, checking last 5 builds is ridiculous. We rather look at the last 400 builds."* — Function Tester

A systems developer added:

> *"Why not use the last 50 instead of 5, and you can filter out the noise and find some trend. You can find two channels, or you can see a slow trend that is going down. However, the last five is not really relevant."* — Systems Developer

Taking more than the previous five test results into consideration could provide more capabilities and used in a way to warn testers in case of unexpected regressions with the indicators. Using more historical data can help to make sense of the existing test result data and reduce the amount of false verdicts.

7. DISCUSSION

The aim of this study was to identify challenges regarding automation of performance regression test analysis and suggest possible improvements for industrial applications. By this, we add to the existing body of knowledge, which especially lacks empirical studies on the interplay of performance regression testing in the context of continuous integration.

With respect to RQ1 (*challenges regarding automation of performance regression test analysis*), the results include:

1. Infrastructure has to be set up in a way that encourages and makes it easier to automate the process. This is especially the case in the context of continuous integration, where it is difficult to assess the test results as they become inadequate quickly.

2. Unclear goals and insufficient requirements slow down the automation process. We did not experience consensus about the concepts regarding performance regression test analysis within industrial practice.

3. The verdict tool used at the case company is a simple, useful tool – but it has serious limitations. Lack of intelligence and usability do not allow to achieve its full potential.

4. Data produced during test execution can be unreliable which makes the automation process difficult. Due to variations of different hardware that are used, successfully evaluating a test case is challenging.

The findings regarding RQ2 (*suggested automated performance regression test analysis improvements*), the most important results are:

1. Goals have to be stated, along with their rationales. As the first step towards full automation, testers need to understand the reasoning behind the goals to be more dynamic.

2. A more centralized infrastructure with clear standards can help the developers and testers to make full use of their tools.

3. A testing map could improve the understanding regarding the connections between each build and new developments. This in turn can help a better assessment of the test results.

4. Tools for the automation process have to be designed in a way that allows testers to be more dynamic and flexible. An early alarm system that includes warning as a result apart from pass and fail would help in some cases, such as core dumps.

5. An intelligent component is necessary for the next step in the automation process. Tools have to make decisions on their own and it is only possible with an intelligent component.

6. Making use of historical test data is highly critical as in order to make better decisions and have more accurate verdicts tools have to take advantage of a rich source of historical data.

7.1 Implications for practitioners

This study presents a set of challenges regarding automation of performance regression analysis in a large company and proposes improvement suggestions for these encountered challenges. Prior to this study, the case company was not confident in verdict system's performance. The verdict system was developed without extensive research simply to reduce some of the manual work for the system testers during the analysis phase. The results of this study helped the case company to make an objective assessment of the verdict system. Four different categories were highlighted as problematic areas and improvement suggestions to overcome these problems were proposed. This study presented a broader understanding of their tools and processes for the case company and helped them evaluate their solution. We believe that based on this, the analysis process can be automated even further and a faster and more accurate analysis phase can be achieved. Testers can control a greater extend of the test cases, thus ensuring a higher level of quality for the product. Developers can avoid investigating unconfirmed bugs and regressions and only focus on debugging and optimizing the code they are responsible of.

We found that the verdict system is a highly useful tool for testers and helps eliminating mundane work within the department and each interviewee from the case department has agreed that having verdict system makes sense for the company and in some cases it would have been impossible to achieve the department goals.

We believe that other companies will find our findings valuable when considering test verdict automation.

7.2 Validity Threats

In order to limit threats to validity, we relied on triangulation between qualitative and quantitative data collection methods, such as interviews, system analysis, workshops and documents study. Furthermore, we informed interviewees about the study beforehand and made sure to give them a general idea about the research. We tested our interview guide with pilot interviews both with academic and industrial experts (at the case company). In order to prevent eliciting false and unreliable information, we guaranteed anonymity and confidentiality of the interviewees. After the interviews were completed, we shared notes with the interviewees and asked them to confirm them.

We specifically conducted interviews with experts from other departments within the case company as well as with organizations to ensure that the concepts observed at the case company are not specific to the company or department.

Based on regular meetings among the authors (which include industrial experts), we made sure that our conclusions are accurate and present in the data.

8. CONCLUSION

In this paper we reported on a qualitative case study on performance regression testing. We found that practitioners find it challenging to efficiently derive verdicts from such non-functional testing in the context of continuous integration and continuous deployment, which implies continuous evolution of software-intense systems. Automation is possible but challenging. We report on challenges and suggest improvements. Based on our results, we encourage more research in automation of non-functional regression testing, especially for embedded, complex systems. This can enable shorter time to market, quicker feedback to developers, and better overall quality of software intense products.

Acknowledgements

We thank all interviewees for their rich input. This work was partially funded by Software Center (software-center.se), a research network between major Scandinavian ICT companies and universities, and the Vinnova FFI projects Next Generation Electronic Architecture and Next Generation Electronic Architecture step 2.

9. REFERENCES

[1] E. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. The oracle problem in software testing: A survey. *Software Engineering, IEEE Transactions on*, 41(5):507–525, 2015.

[2] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 141–150. ACM, 2009.

[3] E. Bjarnason, P. Runeson, M. Borg, M. Unterkalmsteiner, E. Engström, B. Regnell, G. Sabaliauskaite, A. Loconsole, T. Gorschek, and R. Feldt. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering*, 19(6):1809–1855, 2014.

[4] J. W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches.* Sage publications, 2013.

[5] G. Denaro, A. Polini, and W. Emmerich. Early performance testing of distributed software applications. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 94–103. ACM, 2004.

[6] P. M. Duvall, S. Matyas, and A. Glover. *Continuous integration: improving software quality and reducing risk.* Pearson Education, 2007.

[7] E. Engström, R. Feldt, and R. Torkar. Indirect effects in evidential assessment: a case study on regression test technology adoption. In *Proc. of the 2nd Int. WS on Evidential Assessment of Software Technologies*, page 15–20, 2012.

[8] Ericsson.com. Evolved packet gateway, 2015.

[9] Ericsson.com. Traffic and market report, 2015.

[10] M. Felderer and I. Schieferdecker. A taxonomy of risk-based testing. *Int J Softw Tools Technol Transfer*, 16:559–568.

[11] K. C. Foo, Z. M. J. Jiang, B. Adams, A. E. Hassan, Y. Zou, and P. Flora. An industrial case study on the automated detection of performance regressions in heterogeneous environments. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 159–168, 2015.

[12] R. M. Hierons. Verdict functions in testing with a fault domain or test hypotheses. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 18(4):14, 2009.

[13] A. Ieshin, M. Gerenko, and V. Dmitriev. Test automation: Flexible way. In *Software Engineering Conference in Russia (CEE-SECR), 2009 5th Central and Eastern European*, pages 249–252. IEEE, 2009.

[14] E. Knauss, M. Staron, W. Meding, O. Söder, A. Nilsson, and M. Castell. Supporting continuous integration by code-churn based test selection. In *Proceedings of 2nd International Workshop on Rapid and Continuous Software Engeering (RCoSE '15 @ ICSE)*, Florenz, Italy, 2015.

[15] J. Larsson and M. Borg. Revisiting the challenges in aligning re and v&v: Experiences from the public sector. In *Requirements Engineering and Testing (RET), 2014 IEEE 1st International Workshop on*, pages 4–11. IEEE, 2014.

[16] D. L. Morgan. Qualitative content analysis: A guide to paths not taken. *Qualitative Health Research*, 3(1):112–121, 1993.

[17] M. Polo, P. Reales, M. Piattini, and C. Ebert. Test automation. *IEEE software*, (1):84–89, 2013.

[18] D. Stahl and J. Bosch. Modelling continuous integration practice differences in industry software development. *Systems and Software*, 87:48–59, 2014.

[19] E. J. Weyuker and F. I. Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE transactions on software engineering*, (12):1147–1156, 2000.