Faculty of Computer Science
March 22, 2015

TI3706 - Bachelor Seminar

Performance Regression Testing
A developer's perspective

Bachelor Seminar Group

M.A. Hoppenbrouwer	4243889
M.J. Otte	4222695
R.S. Sluis	4088816
R. Vink	4233867

M. Loog	Professor
C. Bezemer	Supervisor

Preface

In the growing world of tools and methods that are available to a software developer, it becomes harder and harder to keep track of which technique is best to use under what circumstances. This holds true for the growing field of performance regression testing, where new testing strategies are being discovered and discussed, and the steps in the process of detecting performance regressions are being improved constantly. This literature study aims to give an overview of these steps in a way that links each step to a point in the software development process corresponding to the challenges that arise when dealing with performance regression testing in practice.

Summary

Contents

List of Figures

1. Introduction

The definition of performance regression testing is: "Performance regression testing detects performance regressions in a system under load. Such regressions refer to situations where software performance degrades compared to previous releases, although the new version behaves correctly."[4] Performance regression testing can be seen as the combination of regression testing and performance testing. "Regression tesing is the retesting of software following modifications."[5] Performance testing is testing performance requirements and specifications of it.[6]
To improve the quality of the software product it could be useful to use performance regression testing in a way the developers can directly improve the quality of the software product. This will be the main focus of this article. There are different aspects in optimizing performance regression tests, which will be discussed throughout this paper. Every aspect will be clarified in the form of a new chapter in this paper.
First of all it needs to be clear how to acquire data and what kind of performance counters will be used for performance regression testing. Next is the filtering of the performance metrics, used to obtain differences in data. The following chapter discusses how to compare the filtered data. Last aspect is that the filtered performance metrics can be reported in the form of visualizations so performance regressions will be detected. At the end of this paper, a research agenda and a conclusion will be provided.

2. Organising the testing process

Different methods of software development exists in software engineering. A different type of development means a different way of testing, so it is important to choose a proper development process before a team starts developing. This chapter explains the development methods and how to organise performance regression testing when using each of these methods.

2.1 Code-and-fix model

"The basic model used during the early days of software devolpment contained two steps: write some code and fix the problems in the code" [7] This 'method' is not appropriate for performance regression testing, because of the poor preparation of testing. Without any idea how the overall design will be, will make the writing of test suites even harder.

2.2 Waterfall method

"The waterfall method establishes a sequence of stages-requirements, specifications, design, coding, testing and maintenance-to guide the development process. "[8]. The sequences of the waterfall method will be done until the development of the system is completed. The waterfall method even became the basis for most software acquisition standards in the past years. [7]. This type of developing has a testing phase. Collins et al researched an agile waterfall process. [9] The research showed that, when using an iterative process, regression testing using the waterfall method can be very effective. This implies direclty that performance regression testing will be effective here, because most of the time performance is the biggest issue in the field. [4]

2.3 Spiral method

"The spiral method creates a risk-driven approach to the software process rather than a primarily document-driven or code-driven process. It incorporates many of the strengths of other models and resolves many of their difficulties."[7] When using the spiral model, the main focus lies on the different risks of a software project. Examples of these risks are: low budget, developing wrong functions or a continous change of functions. These development processes are divided by the property of their risks. So performance regression testing would not be appropriate for the spiral method, because performance regression tests are not seen as risks.

2.4 Scrum

"Scrum is an Agile software development process designed to add energy, focus, clarity, and transparency to project teams developing software systems."[10] A scrum development process is divided into three kinds of phases. Research has been done to show that automated regression tests during the daily builds will greatly improve the quality of the product [11]. This means that scrum is a useful development method for performance regression testing.
There are two organising phases, which are the opening and closing phases and the last phase is the sprint phase which is further divided. The planning and closing phases are organising phases. It is not necessary

to run tests here, because there is either no code (opening) or all the performance regression tests have been executed and passed (closing). Though, planning when to test is an example of an organising aspect which can be done during the opening phase and verifying that the test suite covers all the code can be done during the closing phase.
The actual creating of tests will be done during the sprints. The sprints consist of developing, wrapping, reviewing and adjusing. The wrapping part of the sprint, combines all the implemented code. This process of wrapping is a very appropriate moment to test for performance regressions.

2.5 General aspects of test origanization

Despite the fact that the development method is significant, there are some other organising aspects which could make a difference in the way of performance regression testing.
First of all, it is important who will do the testing. Zaparanuks et al. performed a research where one person did all the testing throughout the project. [12] This person is an active member of the team and watches all the members of the developing team. This method provides a way to deal with any issues that could come up during the process. The appointed person will directly try to deal with issues coming up by the members of the developing team. By appointing this person, the quality of the product is watched the whole time during the project, instead of comparison to the other research where performance regression testing is done during the merge phase. This research proved that the quality of the development got a much higher average score compared to other similar development products. So appointing one tester who only has the task to watch over the quality by testing and for our sake performing regression tests, has an improved effect on the overall process.

Another organising aspect is to make sure a proper test suite is written before programming, and that this test suite is maintained during the project. Test suites should be made with some conditions in mind. Writing complete test suites for complex systems will require the regression testing to run for a long time. [5] So the developers have to decide to either test the complete system, or to write selective test suites. These test suites will only test the main functionality of the system.
After deciding the organising aspects, the actual implementing will be initiated. From now on performance regression testing will be done throughout the process. The developers will decide what kind of data of the implementation will be tested. Performance counters can be used to help decide which data the tests will cover.

3. Gathering data

Once the testing process has been organised to facilitate the particular software development method that is being used, the testing can finally begin. The next challenge that presents itself is what to test for. This chapter discusess what information is relevant to the performance of software and how to handle that information in order to obtain the most accurate representation of the underlying performance.

3.1 Types of data: performance counters

Performance counters are used to measure events that occur duing program execution[13]. Examples are: the number of instructions, loads, bandwidth or the number of executed cycles.
"The goal of analyzing performance counters is to detect if a performance regression has occured, where the performance regression has occured, and what causes the regression" [1].
To detect if performance regressions occur, two different tests will be run: a target run and a baseline run. The target run is the new test run, where the baseline run is a good past run. On both tests equivalent performance counters can be used as input to detect if performance regressions have occured by comparing the output of the tests. Comparing the output of the tests is a big challenge, especially in big complex software systems.

3.2 Tracing

If it is known that a performance regression has occured, the next problem is to trace the performance regression back to the responsible piece of code. Big software systems consist of many types of components, and each component may have many instances. Detecting where the performance regressions come from is very difficult, because there are many performance counters for each instance of each component. This can be very time consuming.
After determining the location of the performance regressions, the cause of the performance regressions needs to be determined. "Understanding the kind of problem usually requires static analysis of source code" [1].

3.3 Accuracy of performance metrics

The fact that performance counters are often used by application developers shows that performance counters are very useful. However, performance counters are not that accurate. Tests have shown some factors can increase or decrease the accuracy of performance counters.
For instance, the measurement error possibly increases as the number of registers increases too. However, this is dependent on the counter configuration and the interface used: experiments with two different interfaces and two different counter configurations show different results, and for only one of the four combinations of the interface and counter configuration the measurement error increases[14]. So most of the times the number of registers does not matter.
Also, the type of infrastructure is very influential. The measurement error reduces a lot when a low-level infrastructure is used, instead of a high-level infrastructure[14]. On top of that, for the user+kernel mode the measurement error gets bigger if the duration of the benchmark gets longer[14]. This means that in order to make the measurement error smaller, less loop iterations should be used. For the user mode the

duration of the benchmark does not matter. This is because of interrupts, that only occur in kernel event counts. More interrupt-related instructions will include if the duration of a measurement takes longer.

4. Data filtering

This chapter describes how the results of performance regression testing can be filtered. This means that out of all results only the results of the relevant perfomance counters remain, so that the developers have a better overview of what to change. This can be done manually, but this costs a lot of time. In this chapter two different techniques are explained: control charts and association rules.

4.1 Control charts

Detecting if performance regression has occured manually costs a lot of time. Comparing each performance counter with a prior good test is a lot of work. A possible way to analyze the results faster is to use a control chart. "The goal of control charts is to automatically determine if a deviation in a process is due to common causes, e.g., input fluctuation, or due to special causes, e.g., defects" [1]. Control charts are often used to detect problems in manufacturing processes where raw materials are inputs and the completed products are outputs. An example of a control chart can be seen in figure 4.1.

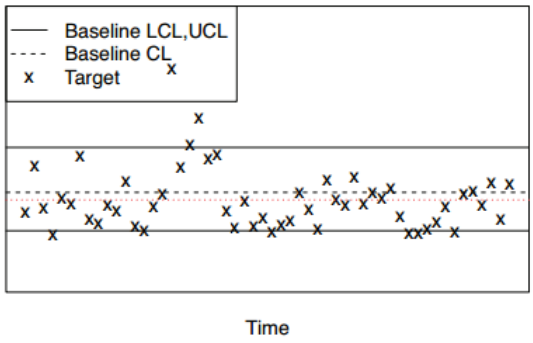

Figure 4.1: from "Example of a control chart"[1]

A control chart consists of a baseline data set (CL) and a target data set (Target). A baseline data set contains the results of the prior test. Based on this data set an upper (UCL) and lower limit (LCL) will be decided. The target data set contains the results of the new test.
From these two data sets the violation ratio can be calculated. This is the percentage of the amount of targets outside the upper and lower limit. If the violation ratio is too high, a performance regression might have occured. In figure 1 only a few targets are outside the upper and lower limit, so most probably no regression has occured.
It now seems easy to detect if performance regression occurs: a certain threshold on the maximum allowed violation ratio can be chosen, and if the violation ratio is higher than this threshold, performance regression has occurred. Unfortunately, this is not the case. There are some reasons that make it hard to detect performance regression. If the violation ratio is low, the probability that performance regression has occurred is low as well. A high violation ratio does not always mean that performance regression has occurred. Some

performance counters are inconsistent, and show different results on the same input data. Because of this, it is possible that the violation ratio is higher than the chosen threshold but no performance regression has occurred. To avoid this, the prior test will be executed again after the new test.

4.2 Association rules

Another way to filter the results of performance regression testing is by using association rules. An association rule has a premise and a consequent. A rule predicts the occurence of a consequent, based on a premise. Association rules can be derived from a frequent item set. "A frequent item set describes a set of metrics that appear together frequently"[4]. To discover a frequent item set, the Apriori algorithm can be used. The Apriori algorithm uses support and confidence to reduce the amount of candidate rules generated: "Support is the frequency at which all items in an association rule are observed together" [4]. If the support is low, the assocation should not be analyzed, but if the support is high it should. Confidence is the probability that the assocation rule's premise leads to the consequent and has a value between 0 and 1. If the value is 0, it means that the confidence has not changed for a new test, and if the value is 1 the confidence of a rule is totally different. A threshold is created to filter out the consequences of the rules that have changed to much.

5. Comparing results

Even though filtering test results can greatly reduce the amount of data that is involved in analyzing testing results, in order to detect performance regressions the results still have to be compared in some way to the results of a previous run of the test suite. The following are some of the problems that arise when doing so and some of the solutions that exist.

5.1 Load dependency

Performance metrics are dependent on the system load during execution of the test suite. This makes it difficult to straight out compare performance counters against results of previous iterations. Jiang et al present an automated way to abstract load testing data and analyze the behaviour of the system. [15] Even though it is not intented to be used for performance regression testing specifically, it uses performance counters to compare system performance to predefined criteria and as such present a solution to the load dependency of performance counters.

5.2 Comparing against historical data

By normalizing and discretizing the performance counters and correlating the results against a historal dataset Foo et al are able to derive a perforance signature [4] consisting of association rules between performance counters. An example of such a signature could be {Arrival Rate = Medium, CPU utilization = Medium, Throughput = Medium}. In combination with statistical techniques, these association rules enable analysts to automatically flag violations of rules in new tests and detect possible performance regressions with a certain confidence.

5.3 Model based testing

Model based testing leverages the advantage of being load-independent against the tasks of having to keep the model up-to-date. The load-independence also only goes as far as the different load for which the model was solved, which means it is not a viable method for detecting performance regressions when system load behaves in ways that was not anticipated in advance. Mi et al offer a way to derive a performance signature that is based on the latency of a transaction and the utilization of system resources, which are adjusted for the existence of outliers [16].

Ghaith et al build on the principle of load independent testing by using a queueing network to model the underlying system and using historical testing data to tune the model to the expected performance under various workloads. [17] This approach combines a transaction profile, a signature of the system resources utilized by some elementary action of the system, with historical testing data. A transaction profile can be obtained using statistical techniques for determing the service demands based on resource utilization [18] or response time [19].

6. Reporting and drawing conclusions

After acquiring, filtering and comparing the data, it has to be reported. The following chapter will show the reporting of the results of performance regression testing. It will touch on visualization and dectection of the performance regressions.

As mentioned in chapter 2, the choice of performance metric is very important. When making a report, this is what it will be based around. A report based on cpu time will have a different structure than a report based on bandwith. The performance metrics require additional data to be able to show regressions. This additional data can be number of times the performance regression test has been executed or the number of times a particular function has been called. This kind of data can be useful to determine if a performance regression has occured. If due to a new functionality a function is expected to be called twice as much since the last revision, a increase in performance can be expected.

6.1 Visualization

When a report is created, a large amount of data can be useful. Only it is very hard and error sensitive to look over those numbers. Visualization can be used to show the data in a way that might be easier to comprehend. By using visual representations such as a graph, can be used to spot a change in revisions. Just like the different structure for reports there are also different means to visualize results of performance regression testing.
Transaction response time is the total time to process a request using the various system resources [20]. The transaction profile represents the lower bound value of the transaction response time for that transaction, or in other words, it is the transaction response time when the transaction is the only one in the system (no queueing) [2].
The transaction profile can be visualised with a bar graph. By comparing the bar graphs of two revisions of the software, it is possible to detect the regressions.

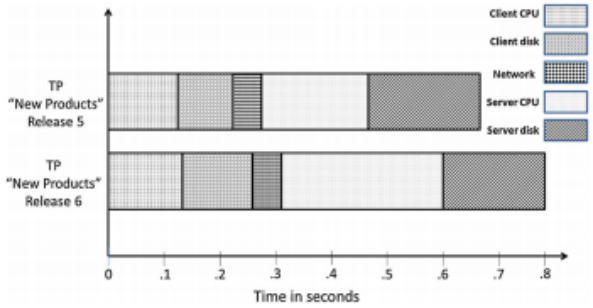

Figure 6.1: from "Anomaly detection in performance regression testing by transaction profile estimation"[2]

The performance regression tests can also be represented in a normal graph. In this type of graph a lot of revisions can be visualized at once. This way there is a better overview of the performance regressions from the entire project and not just two particular revisions. The following graph shows the number of written bytes for each revision of a software project.

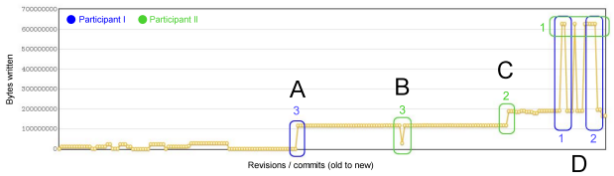

Figure 6.2: from "Detecting and analyzing I/O performance regressions"[3]

These were only two of the many possibilties to visualize results of performance regression testing.

6.2 Detection of performance regressions

When a report and/or a visualization is created any regressions still have to be detected. Detecting regressions is done by analyzing the report and finding a spike in performance. A spike in performance does not always mean there is a regression. It is possible that new functionality is added, which can cause a spike in performance. Looking at the figure above a spike in performance can be seen indicated with an "A". "Phenomenon A: The increase was caused by a bugfix. Before this bugfix, data was not committed to the database"[3].

Detecting regressions is not a discrete process. The measuring of performance is precarious. Because of the error-prone measurements, often a method that provides an accuracy measurement is used. This shows how close to the previous revisions are to the new one. To be more precise with this method it is best to run the tests a many times as possible. This is however impractical, because it could possibly take months or even longer. Because of this, in performace regression testing, the factor of time has to be weighed against the factor of accuracy.

7. Research agenda

8. Conclusion

Bibliography

[1] Thanh HD Nguyen. Using control charts for detecting and understanding performance regressions in large software. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 491–494. IEEE, 2012.

[2] Shadi Ghaith, Miao Wang, Philip Perry, Zhen Ming Jiang, Pat O'Sullivan, and John Murphy. Anomaly detection in performance regression testing by transaction profile estimation. *Software Testing, Verification and Reliability*, 2015.

[3] C Bezemer, E Milon, A Zaidman, and J Pouwelse. Detecting and analyzing i/o performance regressions. *Journal of Software: Evolution and Process*, 26(12):1193–1212, 2014.

[4] King Foo, Zhen Ming Jiang, Bram Adams, Ahmed E Hassan, Ying Zou, and Parminder Flora. Mining performance regression testing repositories for automated performance analysis. In *Quality Software (QSIC), 2010 10th International Conference on*, pages 32–41. IEEE, 2010.

[5] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. Prioritizing test cases for regression testing. *Software Engineering, IEEE Transactions on*, 27(10):929–948, 2001.

[6] Xiang Gan. software performance testing. In *Seminar Paper, University of Helsinki*, pages 26–9. Citeseer, 2006.

[7] Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[8] Kyo C Kang and Leon S Levy. Software methodology in the harsh light of economics. *Information and Software Technology*, 31(5):239–250, 1989.

[9] Eliane Collins and Vincente Lucena. Iterative software testing process for scrum and waterfall projects with open source testing tools experience. *on Testing Software and Systems: Short Papers*, page 115, 2010.

[10] Jeff Sutherland, Anton Viktorov, Jack Blount, and Nikolai Puntikov. Distributed scrum: Agile project management with outsourced development teams. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 274a–274a. IEEE, 2007.

[11] Jeff Sutherland. Future of scrum: Parallel pripelinning of sprints in complex projects. In *Agile Conference*, pages 90–99, 2005.

[12] Jeff Sutherland, Guido Schoonheim, and Mauritz Rijk. Fully distributed scrum: Replicating local productivity and quality with offshore teams. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–8. IEEE, 2009.

[13] W Korn, Patricia J Teller, and G Castillo. Just how accurate are performance counters? In *Performance, Computing, and Communications, 2001. IEEE International Conference on.*, pages 303–310. IEEE, 2001.

[14] Matthias Hauswirth Dmitrijs Zaparanuks, Milan Jovic. Accuracy of performance counter measurements. Technical report, University of Lugano, 2008.

[15] Zhen Ming Jiang. Automated analysis of load testing results. In *Proceedings of the 19th international symposium on Software testing and analysis*, pages 143–146. ACM, 2010.

[16] Ningfang Mi, Ludmila Cherkasova, Kivanc Ozonat, Julie Symons, and Evgenia Smirni. Analysis of application performance and its change via representative application signatures. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 216–223. IEEE, 2008.

[17] Shadi Ghaith, Miao Wang, Philip Perry, and John Murphy. Profile-based, load-independent anomaly detection and analysis in performance regression testing of software systems. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 379–383. IEEE, 2013.

[18] Giuliano Casale, Paolo Cremonesi, and Roberto Turrin. Robust workload estimation in queueing network performance models. In *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, pages 183–187. IEEE, 2008.

[19] Stephan Kraft, Sergio Pacheco-Sanchez, Giuliano Casale, and Stephen Dawson. Estimating service resource consumption from response time measurements. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, page 48. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[20] Raj Jain. *The art of computer systems performance analysis*. John Wiley & Sons, 2008.