**UNIVERSITY OF WATERLOO**

**Faculty of Mathematics**

**THE OPTIMAL PERFORMANCE ENGINEERING ROUTINE**

Veeva Systems

Toronto, Ontario

Prepared by

Gary Jiayi Lin

2B Computer Science

ID 20557560

July 28th, 2016

**MEMORANDUM**

To: Stefan Timofte

From: Gary Lin

Date: July 28th, 2016

Re: Work Report: The Optimal Performance Engineering Routine

As we agreed, I have prepared the enclosed report, “The Optimal Performance Engineering Routine,” for my 2B work report and for the Veeva Toronto Research and Development Teams. This report, the second of four work reports that the Co-operative Education Program requires that I successfully complete as part of my BCS Co-op degree requirements, has not received academic credit.

The Vault Performance team that you lead provides systems performance monitoring, measurements and optimization for both internal software developers and external customers. My job as Intern Performance Engineer required that I develop and execute performance tests, develop data collection/organization scripts and provide overall support for your work. This report is an in-depth study of the optimal schedule for whom and when to develop/execute particular performance tests for a product such as Veeva Vault.

The Faculty of Mathematics requests that you evaluate this report for command of topic and technical content/analysis. Following your assessment, the report, together with your evaluation, will be submitted to the Math Undergrad Office for evaluation on campus by qualified work report markers. The combined marks determine whether the report will receive credit and whether it will be considered for an award.

Thank you for your assistance in preparing this report.

Gary Jiayi Lin (your signature)

Table of Contents

[List of Figures iii](#_Toc459022271)

[Executive Summary iv](#_Toc459022272)

[Introduction 1](#_Toc459022273)

[Analysis 1](#_Toc459022274)

[The Problem 1](#_Toc459022275)

[Emphasis on Architectural Design 3](#_Toc459022276)

[Emphasis on Agile Reinforcement 4](#_Toc459022277)

[Performance Unit Testing 4](#_Toc459022278)

[Performance Regression Testing 5](#_Toc459022279)

[Strategic Disadvantages 5](#_Toc459022280)

[Conclusion 6](#_Toc459022281)

[References 6](#_Toc459022282)

# List of Figures

# Executive Summary

# Introduction

“Problems worthy of attack prove their worth by means of retaliation” stated Danish scientist, inventor and mathematician Piet Hein (Hein, 2). Amongst the thousands of problems software engineers face throughout their careers, software performance quality happens to be one of the most complex and difficult to solve. The struggle to meet software performance targets is comparable to one climbing a ski slope with their skis still on. Also, the target goal, a flag, is positioned halfway up the hill, not at the top. Getting to the flag is difficult enough, as the slipperiness ultimately leads to a long slide down to the bottom of the hill, along with endless frustration. Eventually when one finally makes it to flag, they would discover that they’ve already drained all their energy. Yet, due to the slipperiness and angle of the hill, one would have to use energy simply to maintain the altitude and avoid accidentally rolling to the bottom.

Considering the hill as a measure of software performance, the flag as a foreseeable performance target, the act of climbing the hill as an effort to reach the target, and the energy used as resources invested, the imagery sharpens. Software performance engineering is difficult. It is a problem undoubtedly worthy of attack, and it undoubtedly retaliates.

However, there exists ample tools, strategies, methodologies and practices to ensure that software performance targets are met on a regular basis. This report will evaluate the general problem of poor software performance, the effects this problem has on the software industry, various performance engineering practices and the optimal performance-focused development cycle. Perspectives in this report will alternate between reference to the general software industry and the specific work methodologies I grew accustomed to as a member of the Vault team at Veeva Systems.

# Analysis

## The Problem

When developing software, there really only are two questions of interest. Does this software do what it needs to do? If so, does it do it efficiently? Making software that does exactly what is needed is difficult enough, which is why the majority of efforts and resources are dedicated to ensuring that software simply functions as needed. Thus, often, little or no attention is paid to software performance. However, according to Dr. Williams and Dr. Smith from Performance Engineering Services, “poor performance costs the software industry millions of dollars annually in lost revenue, decreased productivity, increased development and hardware costs, and damaged customer relations” (Williams and Smith, 2002).

The deeper trouble is, even if resources are allotted towards performance operations, they rarely yield a notable difference. Once a system has been implemented atop a rigid architecture, upper-layer performance optimizations are ineffective and inconsequential. They can only offer minute upgrades within individual modules that shave mere milliseconds at a time off of the total run time. Code optimization on a modular scope offers an extremely low effort to performance improvement ratio. In fact, from personal experience optimizing scripts on a module to module basis at Veeva Systems, I judge that return from efforts diminish exponentially, as graphed below.



Figure : Diagram to show diminishing returns in performance as extra labor is employed (Riley, p1).

The difficulty in meeting software performance requirements often stems from early stage architectural design flaws. These flaws are “introduced in early development, but not discovered until late, when they are more difficult and costly to fix” (Williams and Smith, 2002). Though it may seem natural that investing more into the development phase could reduce performance issues in the long run, that is rarely the case. Due to the technical, legal and business complexity of industry-grade software, the end product is almost never an exact manifestation of the original blueprints. Thus, architectural designs rarely accurately forecast future performance issues.

Obviously, software performance issues affect software companies, be it that provide software-as-a-service or software-as-a-product. These issues, if persistent in production, would impact the businesses that buy and use these software, resulting in decreased productivity, decreased revenue and increased expenses. It would also imply weaker relations between software providers and their clients.

## Emphasis on Architectural Design

There exists a plethora of performance testing tools to combat the army of performance issues. As a performance engineer intern at Veeva, I learned to develop and run multiple layers of performance tests: stress tests, load tests, soak tests, component tests, end-to-end use case tests, all of which are significant in their own way. However, these test remain unfruitful if the source architecture, especially in embedded distributed systems, is poorly designed. Thus, the primary focus should lie on software architecture, where extra time, resources and effort should be invested. Of course, no design can be flawless, and certainly not predictive. In fact, with agile movement, development should proceed adaptively as opposed to predictively. As described by the twelfth principle of the Agile Manifesto: “At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly” (Beck et al., 2001). Thus, I propose that at the end of every sprint, after features are delivered, a formal assessment of the architecture be conducted. According to Williams and Smith, one of the best practices for performance assurance involves “performing an architecture assessment to ensure that the software architecture will support performance objectives” (Williams and Smith, 2002). Certainly, this would increase the amount of work required per sprint, which equates to less time for planning, development and testing. Nonetheless, periodical architectural checkpoints in regards to performance would greatly reduce wasted resources towards the later stages of product development, where resources can be allotted to customer satisfaction and request handling.

## Emphasis on Agile Reinforcement

To further tackle performance milestones, one should look to understand the agile development methodology. Agile development works in short two to five-week sprint cycles, where each sprint is structured by four ceremonies: sprint planning, daily stand-up, sprint demo and sprint retrospective (Atlassian, 2016). In respect to meeting performance objectives, there can and should be improvements to all ceremonies of the agile movement.

### Performance Unit Testing

For instance, during sprint planning, product managers should consider previous sprints’ progress and plan accordingly, but at least a quarter of the sprint agenda should be dedicated to performance specific testing. Since, according to Williams and Smith, all developers and managers should be held responsible for maintaining performance standards, I propose that modular performance be a common criterion amongst unit testing (Williams and Smith, 2002). Thus, product managers plan for performance tasks, while developers implement them. Also, having unit testing cover low level performance ensures that daily scrum stand-ups bring more attention to performance. At Veeva Systems, I was often tasked with profiling some feature, then looking through the stack trace to determine which precise method was hogging system time and resources. Should our team implement performance testing as a subsection of unit tests, the need for late stage profiling would diminish, thus freeing up time for more meaningful tasks.

As all performance issues arise from the lower levels of source code, it is imperative that the issue is dealt with at its root. This problem demonstrates the Butterfly Effect, where a butterfly’s wing flap triggers a series of events that lead to a hurricane (Ghys, 2012). An inefficient method that is called several times from multiple sessions will easily build up the server response time. I’ve opened many defect tickets at Veeva pertaining to particular methods that were performance bottlenecks. Though the method itself may only take 500 milliseconds to run, if it is called 4 times when you load a page, that method alone would take up 2 seconds. Therefore, to minimize bottlenecks, I would highly propose increased resource investment into performance unit testing.

### Performance Regression Testing

Moreover, I propose that at the end of every sprint, a round of regression performance tests be run in addition to whatever general system performance tests will be run. A regression test is simply the process of re-testing software that has been modified (Amman and Offut, p.215). On the Veeva Vault team, regression tests are run usually once a sprint, but regression performance tests are run at the end of the release, which is a much longer than it should be. Running regression performance tests allows for frequent updates on end-to-end component performance measures, which may potentially give insight on performance objectives for the following sprint. If Veeva Vault implements performance regression tests per sprint, there would noticeably fewer performance issues when release deadlines approach.

## Disadvantages to Performance Investment

On the contrary, it is imperative to consider the offset of investing resources into performance work. To reserve a significant portion of each sprint to performance efforts implies a loss of time for all other activities. Development deadlines would be tighter, which almost directly correlates to decrease in code quality and, thus, more bugs. There would be noticeably less time for writing and running tests, which means more bugs would eventually slip through to production.

Furthermore, from personal experience at Veeva Systems and job-hunting on Jobmine, there is currently little demand for performance engineers in the software industry. This indirectly decreases the number of qualified software performance engineers, which complicates the hiring process. More resources would be used up in finding developers that have also had sufficient training or knowledge of software performance.

# Conclusion

The search for a perfect performance engineering routine is only a sub-task of a far more sought after answer: how does one create perfect software? There does not exist, at least not yet, an optimal software development schedule. In fact, there may never be. Software developers cannot write perfect code, so to offset imperfect code, significant resources and efforts are invested into quality assurance; armies of test cases, hours and days spent re-planning sprints, and endless knowledge acquisition sessions. Amongst the vast efforts of quality assurance, the commonly overlooked, yet imperative, vision of superb software performance is one of the most difficult to achieve. To optimize corporate performance engineering, I have described in this report in detail a list of considerations and practices. Specifically, during the planning and design phase, there should be a heavy focus on architecture, with thorough technical attention for scalability and responsiveness. I also stress that throughout the periodic sprint cycle, performance-focused sprint planning is carried out, performance unit and regression tests are implemented and that sprint retrospection involves performance progress.

# References

"A Brief Introduction to Scrum." Atlassian. N.p., 2016. Web. 09 Aug. 2016.

<https://www.atlassian.com/agile/scrum>.

Ammann, Paul, and Jeff Offutt. Introduction to Software Testing. New York: Cambridge UP,

2008. Print.

Beck, Kent, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin

Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. "Manifesto for Agile Software Development." Manifesto for Agile Software Development. N.p., 2001. Web. 02 Aug. 2016. <http://www.agilemanifesto.org/>.

Ghys, Etienne. The Butterfly Effect (n.d.): n. pag. ENS DE LYON. 8 July 2012. Web. 3 Aug.

2016. <http://perso.ens-lyon.fr/ghys/articles/butterflyeffect.pdf>.

Hein, Piet. Grooks. N.p.: n.p., 1996. Print.

Riley, Geoff. "Production Function in the Short Run | Economics." The Exam Performance

Specialists. Tutor2u, n.d. Web. 21 Aug. 2016. <http://www.tutor2u.net/economics/reference/production-function-in-the-short-run>.

Williams, Lloyd G., Ph.D., and Connie U. Smith, Ph.D. "Software Performance

Engineering." Best Practices in Software Measurement (2003): n. pag. Www.perfeng.com. Performance Engineering Services, 2003. Web. 2 Aug. 2016. <http://www.perfeng.com/papers/bestprac.pdf>.

Williams, Lloyd G., Ph.D., and Connie U. Smith, Ph.D. "Solving Common Java EE Performance

Problems." 5 Steps to Solving Software Performance Problems (2002): 351-

72. Perfeng.com. L&S Computer Technology Inc., 2002. Web. 30 July 2016. <http://www.perfeng.com/papers/step5.pdf>.

Woodside, Murray, Greg Franks, and Dorina C. Petriu. Sce.carleton.ca. Carleton University, 8

Feb. 2007. Web. 2 Aug. 2016. <ftp://ftp.sce.carleton.ca/pub/cmw/07/woodside-performance.pdf>.