**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](#_Toc458691946)

[Executive Summary iv](#_Toc458691947)

[Introduction 1](#_Toc458691948)

[Analysis 1](#_Toc458691949)

[The Problem 1](#_Toc458691950)

[The Solution 3](#_Toc458691951)

[References 5](#_Toc458691952)

# List of Figures

# Executive Summary

Asdfasdf

Ads

Fa

Dsf

Afds

A

Dsf

Asdfasdfadsf

Adsf

As

Df

As

Df

Asdf

Asdf

A

Dsfasdfas

Dfadsf

Ads

F

Ads

Fasdf

# Introduction

“Problems worthy of attack prove their worth by hitting back” stated Danish scientist, inventor and mathematician Piet Hein, and one would find it difficult to disagree (Hein, 2). Amongst the tens of thousands of problems software engineers face during their careers, performance quality happens to be one of them. It also happens to be one of the most difficult to solve. The struggle to meet software performance targets is never smooth, and often involves remodeling the architecture of at least one major module.

However, there exists ample tools, strategies, methodologies and practices to ensure that software architecture is designed with scalability and responsiveness as core values. This report focuses on determining the optimal types of performance tests to run, when to run them and who to run them. The report also defines and analyses a particular software development cycle that emphasizes assurance of performance quality. 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 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 the answer to the first question is yes. Thus, it is often the case that little or no attention is paid to software performance. However, “poor performance costs the software industry millions of dollars annually in lost revenue, decreased productivity, increased development and hardware costs, and damaged customer relations”, according to Dr. Williams and Dr. Smith (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 with a rigid architecture, performance optimizations can do very little, offering only minor changes within individual modules, shaving milliseconds at a time off 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 unit to unit basis at Veeva Systems, I can graph with certainty the relation between effort and improvement in performance.

INSERT GRAPH

The difficulty in meeting software performance requirements often stems from early stage design flaws in architecture. 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, it might not be 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-as-a-service and software-as-a-product companies. Furthermore, 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.

## The Solution

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 tests mean nothing if the original architecture, especially in embedded distributed systems, is poorly designed. Thus, the primary focus is on the 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 round of performance testing is executed, including assessment of the architecture. 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.

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.

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 all developers and managers should be held responsible for maintaining performance standards (Williams and Smith, 2002), I propose that modular performance be a common criterion amongst unit testing. 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.

Additionally, as all performance issues arise from the lower levels of source code, it is imperative that the issue is dealt with at the root. This problem is an example of 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.

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.

# References

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

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

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.

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>.