

The Case for Early Detection of Performance Bugs, using DSI's CPM Toolkit.

December 18, 2008

Prepared by:

Brendan Lawlor
Process Architect
DeCare Systems Ireland

Tel +353 21 4925 158
blawlor@decaresystems.ie

TABLE OF CONTENTS

EXECUTIVE SUMMARY	3
The Nature of Performance Bugs	3
The Cost of Performance Bugs	4
Early Detection of Performance Bugs	5
About CPM Toolkit	7
About DeCare Systems Ireland	7

PROPRRIETARY RIGHTS NOTICE

All Rights Reserved. This material contains the valuable properties and trade secrets of DeCare Systems Ireland Ltd. No part of this material may be reproduced or transmitted in any form or by any means, electronic, mechanical or otherwise, including photocopying and recording, or in connection with any information storage or retrieval system without the permission of a Director or Company Secretary of DSI Ltd.

EXECUTIVE SUMMARY

The nature of performance-related bugs is such that they incur disproportionately high costs if left undetected. This paper details the particular nature of such bugs, explains their high cost, and describes how this potentially expensive problem can be solved relatively cheaply by the simple expedient of detecting performance problems earlier in the development cycle. This can be achieved by applying Continuous Integration to monitor performance over time.

The Nature of Performance Bugs

Performance bugs are not like feature bugs. They exhibit a number of properties that differentiate them from other types of bugs and require us to treat them differently. These properties combine to make performance bugs

1. Difficult to find
2. Expensive to fix
3. Disproportionately damaging to the application.

Performance Bugs are Deep

Performance issues have the unique property of merging into each other as individual cases arise. That is to say performance problems, by the time they are detected, are perceived as one problem: an underperforming system. The expense of separating this out into individual root causes is an additional expense, over and above the cost of a normal feature bug. The later a performance problem is detected, the greater the chance that it is in fact a complex combination of multiple root causes. Every developer knows that such 'deep' bugs take longer to track down, and longer to fix.

Performance Bugs are not 'Mistakes'

Feature bugs are relatively easy to identify, either through functional testing or code review. Performance bugs don't yield to this kind of analysis because very often they are caused not by a coding error as such, but by a poor choice of design alternatives, or an imperfect knowledge of the underlying behaviour of the programming language being used. The standard mechanisms we put in place to trap feature bugs do not help us when it comes to performance bugs.

The 'Creep Effect' in System Performance

Whereas feature bugs appear suddenly, and disappear when fixed, performance bugs are less clear cut. Because they can often build up over time (for reasons explained previously), they can be harder to detect. A prerequisite for any system that is designed to detect performance bugs is the ability to monitor performance over time, and notify of possible problems when trends are broken, rather than when specific tests fail.

Performance Bugs are Highly Visible to the End User

If performance issues often go unnoticed by development staff thanks to the 'Creep Effect', the same cannot be said of the impact these issues have on the end user. System performance has a disproportional effect on the way an application is perceived by a customer, acting almost at an emotional level. Fully-functional

applications that are perceived to be unreasonably slow can leave users feeling unsatisfied. Fast applications which are not feature-complete, on the other hand, inspire confidence both in the system itself and the team that created it.

These four unique aspects of performance bugs come together to make them almost invisible to normal testing, damaging beyond their relatively small number, and too expensive to ignore.

The Cost of Late Identification of Performance Bugs

A company's reputation is not a number, and the damage that can be done by delivering under-performing systems cannot be measured here. But the costs of project overruns incurred due to the *late identification* of performance issues *can* be counted. In DeCare Systems Ireland, we have been using Continuous Integration for a number of years and have arrived at a way of measuring the savings that the use of Continuous Integration has brought to finding and fixing feature bugs. I will review these calculations below before considering how Continuous Integration can be applied to performance issues, and what the saving would be for this very different class of bug.

Calculation of CI Savings for Feature Bugs

This simple calculation of time saved by using CI makes the following assumptions:

1. If a bug can be found and fixed in 1 hour during construction-time unit testing, the relative cost of finding and fixing it later is

- 250% more during development system testing
- 400% more during QA
- 600% more during Production

(These take into account response time, communication overhead, diagnosis of problem, and knock-on changes forced on other code).

2. Of bugs not found during construction

- 45% are found in Development System Testing
- 45% are found in QA
- 10% are found in Production

This gives the following result:

$$1 \text{ hr in Dev} = (0.45 * 2.5) + (0.45 * 4) + (0.1 * 6) = 3.5 \text{ hrs after Dev}$$

or stated another way:

Every person-day spent fixing feature bugs caught by Continuous Integration represents a saving of 2.5 person days.

Calculation of CI Savings for Performance Bugs

As explained in the first section of this paper, the costs associated with both finding and fixing performance bugs are high compared with feature bugs, and those costs are particularly compounded when performance issues are left to creep and merge over time. Reusing the formula for CI feature bug savings above, but applying higher relative costs and relative incidences (even conservatively higher¹), we arrive at the following result:

$$1 \text{ hr in Dev} = (0.35 * 3.5) + (0.35 * 6) + (0.3 * 9) = 6.025 \text{ hrs after Dev}$$

or stated another way:

*Every person-day spent fixing performance bugs caught by Continuous Integration represents a saving of **about 5 person days**.*

ROI of Early Detection of Performance Bugs

Until recently, the cost of attempting to catch performance bugs early in development has been prohibitively expensive. Normal performance checking, even using the best available tools, is a labour-intensive activity. Because of this, performance checking is typically only performed once the problem had become obvious. As mentioned above, by that point the costs involved in fixing them is many times higher. It seems like a Catch-22 situation. The costs and benefits of catching performance bugs early seem to cancel each other out.

Correct Application of Continuous Integration

However, by *correctly* applying the simple principles of Continuous Integration to the job of performance management, we can drastically reduce the cost of early detection. Continuous Integration automates the search for bugs by building and running unit tests against the latest version of the codebase, and reporting back when there is a failed build or test. Keeping in mind the unique qualities of performance bugs, we can work out what the correct approach to CI is:

Harness existing unit tests by analysing their performance and report on the trend over time.

Setting a simple performance threshold for a given test would take too much time, would require too much maintenance, and in the end would be quite arbitrary. Trending the *actual* performance of a test, and reporting on deviations from that trend, is the correct approach, given the properties of performance bugs as outlined earlier.

¹ 350% more during development system testing;
600% more during QA;
900% more during Production;
35% found in Development System Testing;
35% found in QA;
30% found in Production

Expected Savings of CI for Performance

Assuming a two hour fix for every performance bug caught early, we can say that:

*Every performance bug caught by Continuous Integration represents a saving of **about 1.25 person days.***

The number of performance bugs likely to be found in any given project, depends on the size of the project (measured for convenience though the number of developers), the length of the construction phase of the project, and the test coverage of that project.

The number of bugs that this technique is likely to find depends on a number of factors. We can generalize these factors into the following formula:

Bug Rate (Bugs Created per Person Day) x Total No of Person Days x Test Coverage / 100

Even for a very modest coverage of say 10%, and assuming that a developer creates a performance bug every 10 days, then in the course of a 3 month, 3 person project, this technique will yield 1.8 bugs, saving 2.25 person days (or 1.25% of the total project effort).

Greater coverage will naturally yield a greater return. For example a 100% test coverage will yield 18 bugs, saving 22.5 person days (12.5% of total project effort).

About CPM Toolkit

The CPM Toolkit introduces Continuous Performance Management into a Continuous Integration environment. CPM Toolkit automates the collection of performance, memory usage and code coverage data to generate performance benchmarks, while providing project management level visibility across development projects.

- Combine the power of JProbe with a *Continuous Integration (CI)* environment to incorporate cost effective performance management early in the development lifecycle.
- Establish *performance benchmarks* and easily track performance deviations from build to build.
- Prioritise performance monitoring across the entire *development team* by extending the results of JProbes' Analysis Engines to the project team.
- Analyse performance behaviour *throughout the development phase* to establish normal patterns of behaviour and build a performance profile of your application.
- Implement a "*find early, fix early*" practice to prevent expensive and time consuming performance tuning in production.
- Provide a greater level of *confidence* prior to moving code into system test, and subsequently user acceptance testing by removing performance issues at the development stage.
- Gain *project management level visibility* across all projects under development using the CPM-at-a-glance dashboard.

About DeCare Systems Ireland

DeCare Systems Ireland (DSI) was established in 1998 and is a subsidiary of DeCare Dental LLC, one of the largest dental benefit management companies in the United States. DSI is an enterprise software development company specialising in architecting, developing and integrating custom .Net and Java applications. With over 140 software technology professionals on staff, DSI provides unique, customer-specific, cost-efficient solutions for clients including Amazon, Avon, Expedia and a number of large US healthcare insurance carriers. Serving both private and public sector organisations, DSI focuses on Architecture Blueprinting, eSolutions, Bespoke Application Development and Application Performance Management.

DeCare Systems Ireland Ltd
Building 1
University Technology Centre
Curraheen Road
Cork
Ireland

Phone: +353 21 4925 100 | Email: info@decaresystems.com | Web: <http://www.decaresystems.com> | Blog: <http://blog.decaresystems.com>