

The Case for Continuous Performance Management in Maintenance Projects

February 24, 2009

Prepared by:

Brendan Lawlor
Process Architect
DeCare Systems Ireland

Tel +353 21 4925 158
blawlor@decaresystems.ie

TABLE OF CONTENTS

EXECUTIVE SUMMARY	3
The Importance of Coverage.....	3
Even Simple Unit Tests Give Real Coverage	3
About CPM Toolkit	6
About DeCare Systems Ireland	6

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

In previous white papers¹, we have made the case for the importance of Continuous Performance Management, and the cost-saving opportunities that CPM presents. This paper concentrates on how those savings can be realised in projects that have moved into production and are being maintained – even those projects whose test coverage is very low. Auto generated unit tests are cheap to produce but normally don't offer much *test* value. What they *do* offer is a high degree of test *coverage* and they are ideal vehicles for monitoring performance.

The Importance of Coverage

Continuous Performance Management, as described in our previous papers, monitors the performance over time of existing unit tests. From this it is clear that CPM's usefulness in finding performance issues is proportional to the percentage of the code covered by those unit tests; the higher the coverage, the greater the number of performance issues detected early. But what about projects which have low test coverage, or no coverage at all?

Projects that are already in production can still suffer from performance degradation over time. Maintenance projects that have low test coverage often don't return the considerable investment required to development the missing unit tests. How can Continuous Performance Management be applied to maintenance projects, given the importance of test coverage in CPM? How can the integrity of the codebase be maintained without damaging the integrity of the business case? The answer lies in the fact that while testing is expensive, coverage can be surprisingly cheap.

Even Simple Unit Tests Give Real Coverage

Writing a good unit test requires knowledge of what inputs are likely to properly test a class's intended behaviour, and what values should be returned as outputs in order for the test to pass. It passes if the return values are as expected, and fails if they are not. Knowing what those return values should be, and coding the necessary comparisons into a unit test, is the most expensive and time-consuming part of writing a unit test.

But creating a simplistic unit test – one that merely invokes methods using a variety of input parameters, without much regard for the values returned, is so easy that it can be automated. There are a number of software development tools² that can be used to auto-generate unit test classes for an entire codebase in seconds. The value of tests that are generated in this way is limited, but the price of the tools that provide them is low (in the case of the tool used to prepare this paper, \$100). Crucially, these simplistic tests *do* provide high rates of coverage. And in order to test the performance of a system through its unit tests, coverage is all you need. Remember that the most expensive aspect of creating

¹ See

http://www.decaresystems.ie/downloads/Cutting_Application_Development_Costs_CPM_Toolkit.pdf and
http://www.decaresystems.ie/downloads/CPM_Whitepaper_Case_for_Use_in_New_Application_Development.pdf

² <http://www.parasoft.com>
http://www.agitar.com/solutions/products/automated_junit_generation.html

unit tests is the building-in of expected values, but these expected values are not needed in order to execute performance tests.

CPM piggybacks on the source code execution paths performed by unit tests. It is the time and memory performance of these execution paths that CPM monitors, analyses and reports on. The more expensive test result comparison is simply not required for CPM to perform its task.

Projects that have made it to production without any unit test coverage can dramatically increase their test at low cost. While the value of this coverage is limited in its ability to find feature bugs, it has great potential to help detect performance degradation. The low-cost execution paths made available by unit test generation guarantee a greater detection rate of performance issues over time.

Calculation of CPM Savings on Maintenance Projects

The coverage provided by auto-generated unit tests is unlikely to be 100%, because in order to ensure every line of a method is executed, we often need to be able to control conditional statements by choosing particular input values. But the *increase* in coverage that can be provided by auto-generated unit tests, and thus the increase in the number of performance bugs that can be detected, is very significant.

What follows is a description of generating coverage for a small project of 27 classes, using an inexpensive code generation tool³. The initial line test coverage rate was zero⁴.

The generation tool was applied from within an Eclipse, and the time required per class was about 30 to 60 seconds from start to end. This clearly compares favourably with the time required to hand-craft unit tests, measured in hours (over their lifetime, and including updating and refactoring) rather than seconds.

The classes were of varying types – some were simple bean-like constructs while others were more complex services. There was a reasonable amount of interconnectedness between them so the test generation tool had to create a significant amount of stubbing in order to isolate each class. The generation required each class to be selected individually in the IDE, and for each class, a simple wizard was presented with some simple (and constant) choices to be made each time.

In order to correctly follow the standard project directory structure, each generated test had to be copied out of the package in which it was created (the same as the class under test) and moved to a directory especially for unit tests. When the process was complete, and the test coverage tool was run again, the resulting line test coverage for the 27 classes came to 71%.

To summarise, for an initial investment of \$100 and an ongoing cost of 1 developer-minute per covered class, the result was a line test coverage of over 70%.

From a previous white paper:

³ CoView from Codign Software: <http://www.codign.com>

⁴ As tested by Cobertura, an OSS coverage measurement tool:
<http://cobertura.sourceforge.net/>

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

and

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

The rate of bug detection is proportional to the effort invested in development, and also to the coverage. When the result is expressed as a percentage of the overall effort, it is dependent only on the coverage percentage.

The *cost* of achieving that coverage is *also* proportional to the overall effort (in this case, the amount of code that was produced, and now must be covered), but at a much lower rate. This means that if you generate unit test coverage for your entire codebase⁵, the cost of achieving coverage is always going to be lower than the savings due to increased bug detection. On average, the amount of time required to generate coverage will always be less than the amount of time saved by finding performance issues.

Note that the tool chosen for this test was inexpensive to buy but required hands-on operation and so had a cost associated with the overall size of the codebase. For very large codebases, it may be more economical overall to run more expensive test generation tools that can be run once and without developer intervention.

In the case of a generated test coverage of 70% on a maintenance project, we can reuse the calculations from the previous white papers: The savings given by CPM on such a project would be 8.75% of total maintenance effort.

⁵ For the sake of completeness, it's worth specifying that coverage should be generated for most or all of the codebase. If we look at the other extreme where you invest 1 minute in covering 1 class out of a possible 100, the chances of detecting performance issues in your system are roughly 100 times less than if you invest 100 minutes covering all 100 classes. I.e. you will probably not find any performance issues, and therefore not save any time - although you will only have lost 1 minute of time in achieving coverage.

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>