



One Stop Web Systems Ltd

Phil Hanchet

PO Box 39281

London SE3 9WY

0845 260 1064

[www.OneStopWebSystems.com](http://www.OneStopWebSystems.com)

Management of Outsourced Code Quality

7<sup>th</sup> March 2008

Draft A

*Abstract*

*With a growing tendency to outsource software development, how can you be sure that the result is of sufficient quality, appropriately documented and the system is able to be rebuilt independently? This report details our approach to this important subject.*

### Table of Contents

Document History..... 2  
Introduction..... 3  
What to look for..... 3  
Our Approach..... 4  
Summary..... 5  
Glossary..... 6

### Document History

Version	Author	Date	Notes
Draft A	Phil Hanchet, One Stop Web Systems Ltd	07MAR08	

## Introduction

With a growing trend to outsource software development, how can you be sure that the result is of sufficient quality, able to be easily maintained, that the delivered system and its documentation are consistent, and the code can be understood and rebuilt by subsequent teams, be it internal or external?

The subject is broad-reaching and relies heavily on training and experience (classic software development principles still apply no matter what the technology or approach).

## What to look for

Regardless of whether your system is Open Source (i.e. able to be built from the source code), commercial Off The Shelf (i.e. delivered as compiled system components) or a mixture of the two, there are some vital aspects about delivered systems that are very often overlooked amongst the jubilation of having a running system delivered in time to show the stakeholders;

1. Is the code of sufficient quality, such that it is easily able to be maintained and updated by your in-house team or subsequent third parties?
2. Is the system architecture sensible and intuitive, with appropriate levels of re-use / libraries for future development?
3. Is the documentation sufficient and up to date (i.e. in sync), including;
  - a. System architecture and design
  - b. Functional spec
  - c. UAT
  - d. Deployment
4. Has the code been commented sensibly?
5. Has the code been built to be internationalized easily? (No embedded String literals etc.)?
6. Does the system comply with all necessary standards (not just WAI AA, but also XML standards, HTML, Java etc)?
7. Are the configuration files intuitively located and their content sensibly arranged?
8. If external libraries are used:
  - a. Are they supplied by well-established providers?
  - b. Are they currently supported?
  - c. What is the process if a bug is found in the library?
  - d. For open source libraries:
    - i. Do they compile using this version of the compiler?
    - ii. Is the code of sufficient standard?
9. What level of error, warning and info logging is built into the system and how is it controlled? How are messages 'bubbled up'? How are message logs rolled?
10. What performance monitoring is included in the system? How are the suppliers sure that the code is efficient? Are any tools used (McCabe cyclomatic complexity for example – see glossary)?

11. Does the system fail gracefully under load?
12. Is the code *really* under version control?
13. Are you able to build and deploy the system yourselves (for disaster recovery purposes, or delivery to subsequent developers)?

## Our Approach

The key role here is the **build manager**, an often overlooked role yet it can be the single biggest injection point for quality in any software system. The build manager can be seen as a gate keeper who handles the following responsibilities:

- Acceptance, date-recording and archiving of the batches of code from the provider[s].
- Addition of the new code to the Version Control System including:
  - o All source, documents, images, build scripts and test results.
  - o Major/ Minor version naming
  - o Version to Functionality mapping (i.e. what version does what)
- Clear reporting of the progress of the system development, including highlighting of areas of risk.
- Code review and standardization (including automated formatting), including direct feedback to the developers (i.e. maintenance of agreed standards to ensure they don't slip, particularly as staff leave).
- Verification of system software architecture against the provided documentation.
- **Development, maintenance and test of the automated build and deployment scripts.**
  - o This is the area that takes much of a build managers time, but if done correctly will record all the intricate details of the system's environment. Very often it is not possible to fully automate roll-out, particularly with third party system – but this is where direct communication with the third party supplier's own developers can often help.
  - o Build scripts and deployment scripts are maintained with the relevant code
- Independent research and location of all external libraries to ensure the suppliers are suitable (including building all open source libraries from source).
- Setup, maintenance and deployment to the UAT system.
- Setup, maintenance and deployment to the live system.
- Generation/validation of deployment documentation.
- Patch maintenance.
- Roll back tests using the version control system.
- Backups and off-sites for live, UAT and version control.
- Ghost management.
- Disaster recovery planning.

## Summary

Undeniably, the devil is in the detail here. The main problem you face with outsourced code is that functionality and timescales very often override quality – which is a short term win bringing long-term problems.

Adopting the build manger approach brings the evaluation of quality in-house, as well all allowing multiple external providers to deliver code at the same time, and those providers be objectively assessed on grounds other than cost and deadlines.

The output from the build manager takes the company a long way toward ISO9000 accreditation, as the role is very much about documentation of process – the build manager’s “How do we automate this?” is very close to the ISO evaluator’s “What is the process for this?”.

Additionally, it removes your reliance (and therefore risk) on individual supplier’s in-house standards, who very often do not expect their code ever to be read.

The in-house build manager approach mitigates risk, improves code quality and system reliability and enables multiple teams to work together inside a known framework.

Finally – should a given supplier founder, all is not lost as the project can be picked up from a known state by subsequent third parties.

## Glossary

### Cyclomatic complexity (source: [Wikipedia.org](http://Wikipedia.org))

The cyclomatic complexity of a section of source code is the count of the number of linearly independent paths through the source code. For instance, if the source code contained no decision points such as IF statements or FOR loops, the complexity would be 1, since there is only a single path through the code. If the code had a single IF statement containing a single condition there would be two paths through the code, one path where the IF statement is evaluated as TRUE and one path where the IF statement is evaluated as FALSE.

Cyclomatic complexity is normally calculated by creating a graph of the source code with each line of source code being a node on the graph and arrows between the nodes showing the execution pathways. As some programming languages can be quite terse and compact, a source code statement when developing the graph may actually create several nodes in the graph (for instance, when using the ":" ternary conditional operator in C, C++, C# and Java).

In general, in order to fully test a module all execution paths through the module should be exercised. This implies a module with a high complexity number requires more testing effort than a module with a lower value since the higher complexity number indicates more pathways through the code. This also implies that a module with higher complexity is more difficult for a programmer to understand since the programmer must understand the different pathways and the results of those pathways.

One would also expect that a module with higher complexity would tend to have lower cohesion (less than functional cohesion) than a module with lower complexity. The possible correlation between higher complexity measure with a lower level of cohesion is predicated on a module with more decision points generally implementing more than a single well defined function. However there are certain types of modules that one would expect to have a high complexity number, such as user interface (UI) modules containing source code for data validation and error recovery.

©2008 OneStopWebSystems Ltd