TESTING AS A DRIVER FOR DEVELOPMENT CHANGE

Eurostar 2002

Graham Thomas

✉	**graham@badgerscroft.com**
☎	**0044 (0)7973 387 853**
www	**www.badgerscroft.com**

Abstract

This paper uses as a case study the work carried out by Graham, as a test manager over the last 18 months, starting with identification of the problems, kicking-off a change program, and then details the initiatives that have resulted, including the definition of a test friendly development lifecycle. To add to the complexity the development group have been investigating agile methodologies such as XP.

1 <u>Introduction</u>

This paper uses the work carried that I have been involved in over the last 3 years working initially as a testing manager, and latterly as a development manager with responsibility for quality through-out the development lifecycle and specifically from developer to client. The paper focuses on the last 18 months where the effort has been in driving changes in the development lifecycle from a testing perspective.

This work has been carried out in a small to medium sized financial systems software house which produces solutions for the treasury dealing market place. This is a dynamic, demanding and challenging market place based upon the management of risk, which over time has also come to apply to the software development activities.

The paper is divided into 5 sections and the following table gives a brief overview of their contents:

Section	Contents
Improving the Testing Process	**This section looks at the improvements made to the existing testing processes over the first 18 months of the timeframe. Looking at testing strategy, process, metrics and reporting.**
Analysis of the Results	**This section reviews the effectiveness of the test process improvements.**
Development Change Program	**This section outlines the development change program that arose from the analysis of the test process improvements of the first 18 months.**
Achievements to Date	**This section reports the achievements to date, and gives an outline of the work still to be carried out.**
Where Value was Added	**This section details where value was added, what the success were, and more importantly what the failures where.**

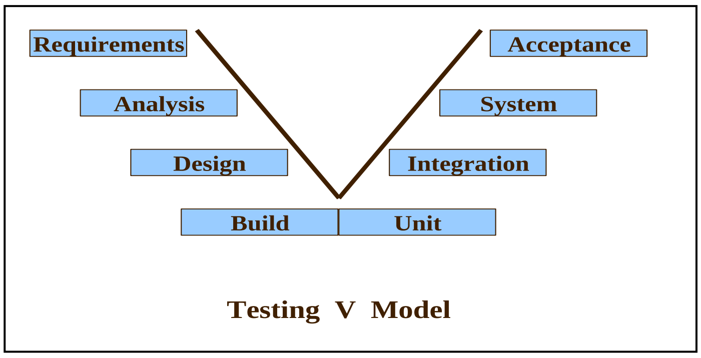
This has been an interesting journey along a twisting and bumpy road, with more than a few surprises along the way.

2 Improving the testing process

The first section of this paper deals with the testing process improvements that were made over the first 18 months.

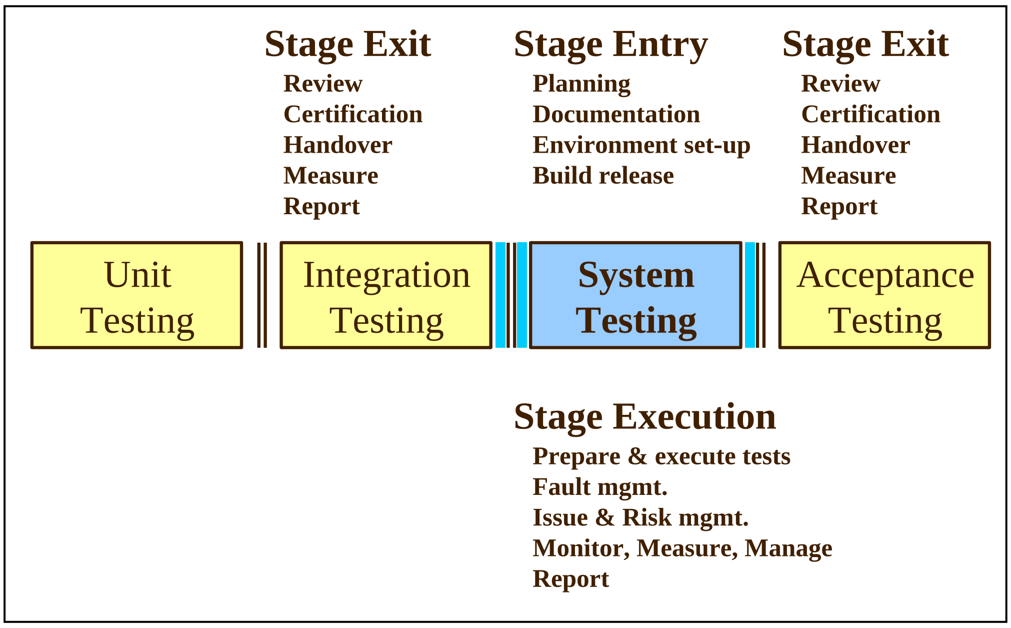
The first step was to produce a formal testing strategy, based upon the V model, which was agreed at board level, and could be articulated to all.

Each stage of the testing lifecycle was clearly defined, and the objectives detailed so that all were clear about their roles and responsibilities.



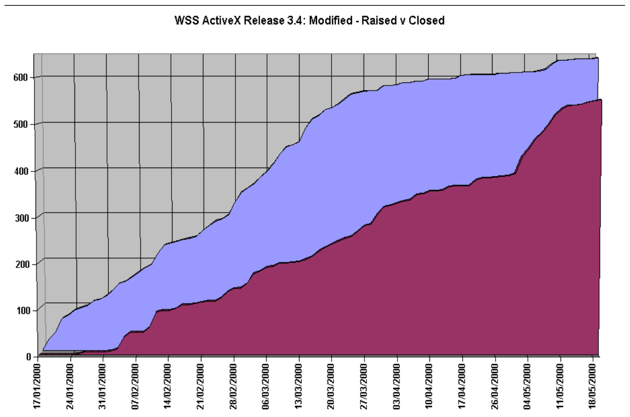
To implement the strategy we defined a set of testing processes, based around the four stage testing lifecycle. For each stage in the lifecycle we defined the following processes:

- **Stage Entry**
  A review checkpoint to see if we are in good shape to carry out the stage.
- **Stage Execution**
  Where the testing activities are carried out, and reported upon.
- **Stage Exit**
  A review checkpoint to ensure see whether we have achieved everything that we set out to do, and have reached the desired quality levels in the testing we have performed.

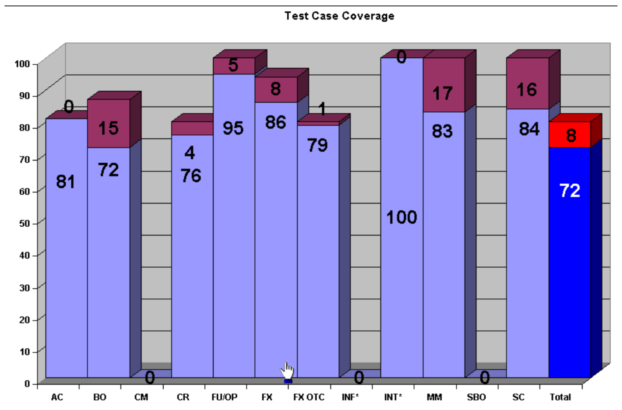


Whilst the Testing Strategy was being prepared and approved the fault recording and reporting process was completely revamped. This allowed the testing team to maintain accurate fault records and produce simple but powerful metrics, e.g.;

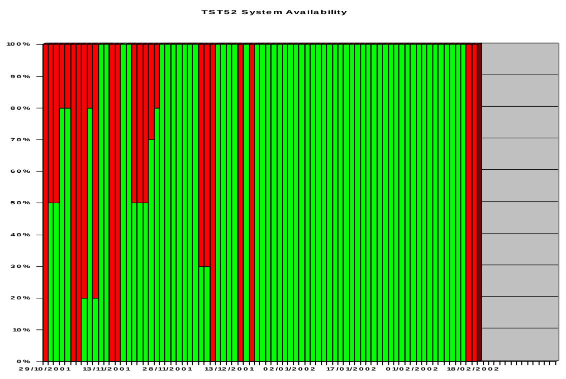
- **Issues Raised versus Issues Closed S Curve**. A simple cumulative fault total graph showing the number of faults raised and closed over time. This can be used over time to track trends and fault fixing progress.



- **Test Coverage**. Showing by product what level of test coverage had been achieved. Test coverage linked back to requirements so an overall quality assessment could be made.



- **Environment Availability**. The testing team were not responsible for the maintenance and support of their own test environment, and this chart was used to report availability of the test environments. This chart uses a simple colour scheme, Green is good and Red is bad. As can be seen, environment provision during the initial stages of a project is often critical.



3 Analysis of the results

After we had been running under the new strategy and improved processes for about 18 months we stepped back and reviewed effectiveness to date. When we looked at the testing stage entry and stage exit quality gates this is what we found:

Stage Entry	
Some Software Components are Delivered Late	The level of testing that we were carrying out, Release Testing, required a complete system delivery. And late software deliveries can lead to delays in the commencement of test execution.
Incomplete Documentation	For some parts of the system we did not receive documentation ready in time for test planning, analysis and preparation. This affected the depth of test planning that we were able to undertake.
Test Environment Availability	This will always be an issue in resource constrained situations, where the testing team do not have dedicated test environments, or where ownership of the environments lies outside the team.
Stage Exit	
High Percentage of Integration Faults	Our metrics helped us to identify that we were experiencing a higher percentage of integration faults than we would have anticipated from a Release Test, considering that the software had already been though a Unit Test and Integration Test stage.
Leading to a Lack of Test Coverage	This was partly a knock on effect from not having planned in depth, and also, because we were spending time detecting and dealing with the integration faults, therefore we did not achieve the desired test coverage in the planned timeframe.
Project Priorities Steer Testing	Real project priorities will inevitably steer the testing effort, especially towards the end of the cycle. This can change the focus of testing, and sometimes lead to inefficient activities.

This analysis led us to the conclusion that although we had improved the testing process, for these improvements to be effective we needed to improve aspects of the development lifecycle as well.

4 Development Change Program

So, early last year (2002) we found ourselves in the position that we had made quite significant improvements to the testing process, but that we were still experiencing problems that were beyond our ability to resolve from within the testing team. At this point we put together a simple roadmap, which grew into a change program.

The reasoning went like this; we need to make changes in a number of areas, Development, Software Management, Testing (because there are always further improvements that can be made), and Programme Office. These areas we came to call streams.

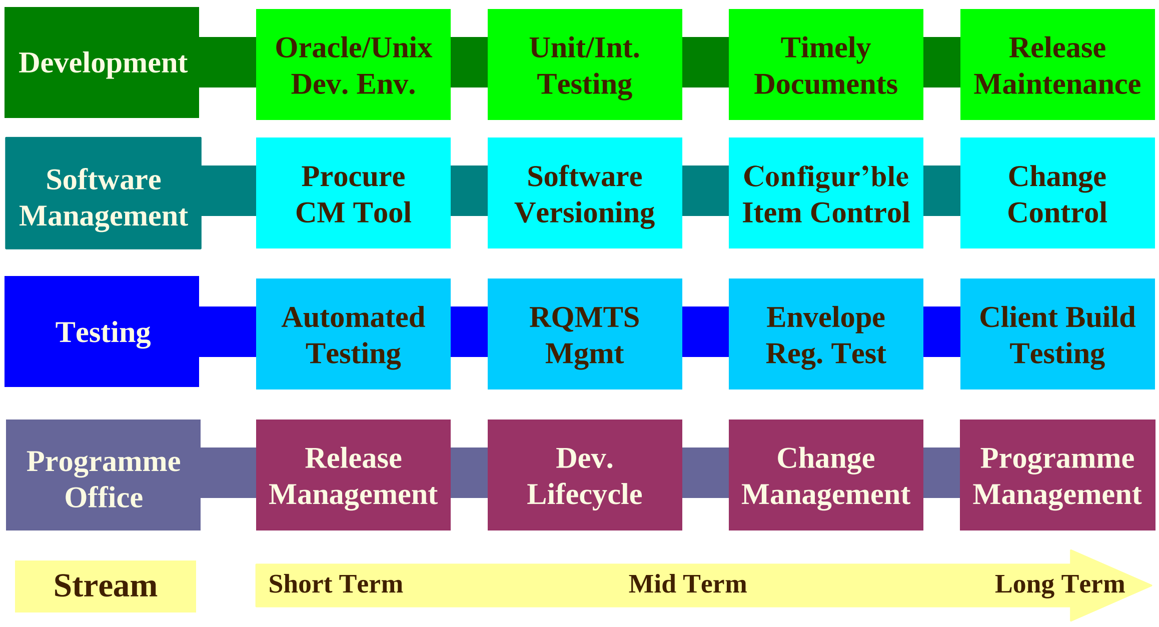
Then we looked at what we needed to achieve, short term to get things going, mid-term to stabilise and develop, and finally long term, where we actually wanted to be. This gave us the following activities:

For Development:	
A development environment which reflected our delivery platform.	This would reduce some of the porting problems that we experience.
Improved Unit and Integration testing	This was an area ripe for tool exploitation.
Timely documentation to drive the testing process	Testers never get enough documentation, nor is it produced soon enough.
The ability to maintain individual variants of releases	This is key when you have 3 platforms, 4 active versions and 10 products.
For Software Management:	
Procure a Configuration Management Tool	We already had in place the VMS Code Management System and Visual Source Safe, but we wanted to move to a single repository, for process and control reasons.
Implement Software Versioning	Understanding the relationship between multiple versions of software.
Introduce Configurable Item control	Control all of our configurable items within the repository.
Deliver Change Control	Use the CM repository to help provide impact analysis for change control.
For Testing	
Deliver Automated Testing	In common with a lot of organisations we were tasked with delivering on the promise of test automation.
Requirements Management	We placed Requirements Management in the testing stream for the short term whilst we determined the right group to own it.
Envelope (interim deliverable) testing	We wanted to be able to regression test interim deliverables using test automation.

Client Build testing (especially important with a highly configurable platform)	Leveraging the Configuration Management activities meant we could test on the client build version of software pre-delivery.
For Programme Office	
Release Management	The ability to manage at a release level.
Development Lifecycle	The definition of a development lifecycle.
Change Management	Change Management, including authorisation and approval.
Programme Management	And finally pulling all of the development activities together under the aegis of Programme Management.

4.1 Change Program RoadMap

We also worked out the dependencies between the activities, and the dependencies between the streams, giving us the change program roadmap below.

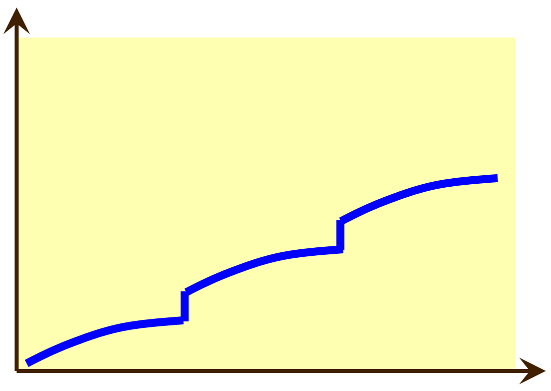


The long term activities show that we were also beginning to position ourselves to be able to respond to requests from the North American customer base for CMM (Capability Maturity Model), Level 3 certification in a time frame of around 18 – 24 months.

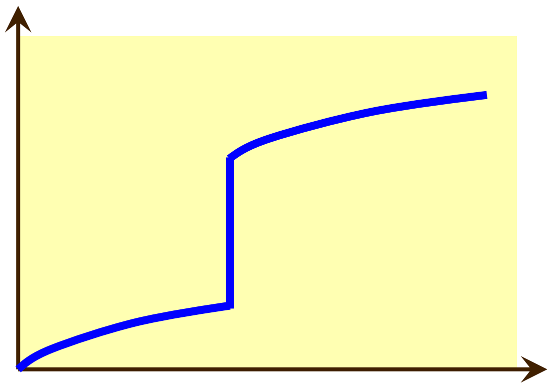
4.2 Change Program Models

We needed to understand how to make that change, so we looked at a number of change program models.

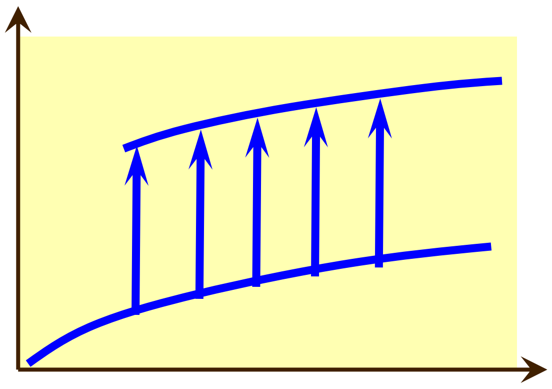
The first model is the **incremental** model. Continuous but small changes made over time. This is why when, for example, you buy a replacement network card for your computer the card is half the size of the one you bought two years ago. This is Continuous Improvement. This model unfortunately doesn't deliver large amounts of change and assumes that the base process is fit for purpose to start with. This approach didn't seem to deliver the amount of change that we were looking for.



The second model, **Step Change**, incorrectly titled in my opinion, is where the organisation takes a dramatic change from one way of working to another, rather like a Big Bang approach. These don't always work, London Ambulance being one of the prime examples. Everything is tipped on its head, people want to revert to the old system when it doesn't work, and there is a lot of resistance. This approach was deemed too risky.



The third model, **Thin Threads**, is an altogether different approach, where you take a single process, and elevate that to the new level that you desire. You have to ensure that your model can work with all of the old processes and with new ones when they also come on stream. This is the approach that we took with our Fault Database. Process change is never too radical, and over time more of the organisation has shifted to the new level.



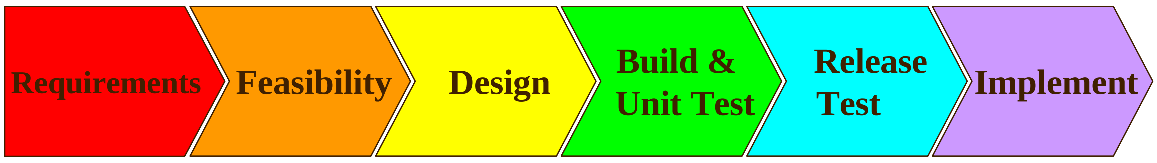
This sounded like it would deliver a high rate of change, with the minimum of disruption so we chose to adopt the Thin Threads approach.

5 Achievements to date

So this time last year we set off on our change program, following our thin threads change model, with our roadmap in our hands.

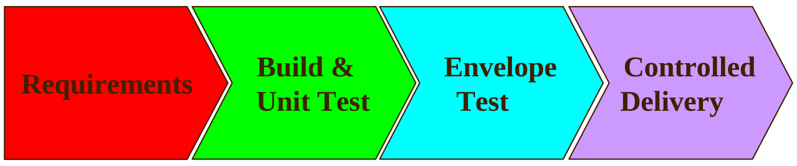
5.1 Development Lifecycle

We defined a development lifecycle. A standard 6 stage structured process, including testing and software delivery. This includes stages for Requirements, Feasibility, Design, Build & Unit Test, Release Test and Implementation. We named this our **Base Development Model**.



We also defined this at the detail level understanding, incorporating all of the processes within each stage of the lifecycle. At the definition level we have also modelled the information flows with the client as well.

We then modified the development lifecycle for small items. This makes up a high percentage of the development workload and we wanted an agile approach to deal with it.



We still start with requirements definition. For example, 'a request from a client to change the sort order on a report'. Due to the nature of the business, most small items that we are dealing with are probably specified in quite some detail and no further feasibility analysis is required. Feasibility therefore becomes an optional stage. The requirements may also include details of the design, so design too can become an optional step. We still have to build and test the code. A full release test is not appropriate for this type of item, so we undertake an envelope test, (which is just the delivery item). In terms of the report we would test that we can generate it in the specified sequence. Finally we replace our implementation process with a controlled software delivery, and hand on to the client for their acceptance.

Thus we have the ability to respond to smaller requirements in an efficient fashion and also understand the process for larger items of work.

5.2 Requirements Process

In terms of requirements we have improved the following:

- We have **tightened up the requirements definition**.
  We have a generated a template for all requirements, enhancements and maintenance items.
- **Testing Specification**
  In the template we request information about how to test the item, including, set-up, configuration and data requirements.
- **Acceptance Criteria**
  We request acceptance criteria so that we can tell prior to delivery whether the item will be acceptable to our client or not. If not we can fix it first.
- **Monitoring**
  We are closely monitoring the process, to ensure that it is working correctly and efficiently. To do this we have 12 separate statuses that an item could be in during the lifecycle.
- **Requirements Driven Development**
  The requirements process now drives the development activity, and work does not commence on any item without the necessary controls and approvals.
- **Clear Targets**
  We have also introduced clearly defined internal targets for workloads and quality threshold.

5.3 Test Automation

Test Automation has seen some key changes over the last year.

- **Automated Test Scripts**
  We have generated automated test scripts for key systems functionality. We have over 500 test cases running automatically, testing core system processing.
- **Multiple Script Executions**
  The automated test suite now runs on multiple PC's, currently 5, controlled though a master terminal.
- **Unattended Running**
  We have put a lot of effort into making sure that the suite can run unattended and overnight. This has meant developing it to cope with unexpected inputs, vary timings with processing loads, and re-start if necessary.
- **High Volume**
  To date we can successfully execute over 8,000 transactions in a single run.
- **Extensible & Scaleable**
  The test automation architecture that we have employed is extensible, allowing us to add new components, and scaleable, allowing us to varying the number of records in the test database driving the scripts.

This really has helped automate and speed up some of the routine aspects of testing, and increased our throughput within individual testing cycles.

5.4 Unit Testing Tools

We have also been looking at Unit Testing Tools to assist the developers. We have been considering a suite of tools which delivers the following benefits:

- **Automatic Static Source Code Analysis**
  The developers have warmed to this feature allowing them to define a set of good practice rules and then test for them.
- **Runtime Error Analysis**
  Development environments are very complex these days and any tools which can help with diagnosis of what actually went wrong are extremely beneficial.
- **Automatic Performance Analysis and Optimisation**
  Performance and optimisation are still as important today as they have ever been; the emphasis has just switched from optimising because of hardware limits, to optimising execution time because there is so much data to process. During the evaluation this was one area that provided very real benefit, optimising a frequently used utility and reducing run-time by 33%. This really helped to sell the benefits of the tools to the developers.
- **Automatic Code Coverage Analysis**
  Code coverage is also beneficial, but I must admit here that in discussions with the developers this is one area they have yet to be convinced about. Mainly because they work within the IDE (Integrated Development Environment), and the performance tools create a separate instrumented executable, which can't be run in the IDE, and doesn't give the developers direct access to the source code.
- **Automatic Error Detection and Recovery**
  This can be quite useful for new developments, although at this point we are unsure whether we are going to retro-fit this to our heritage code.

5.5 Configuration Management

We brought in a Software Manager from a large company, about two years ago, who has Configuration Management experience, and this is now beginning to bear fruit.

- We have identified all of the types of C**onfiguration Items** that we are dealing with throughout the organisation, including; code, environments, infrastructure e.g. Operating system versions, data, configurations, documentation, test scripts, etc. In fact there are 26 types in total.
- We have developed a **Branching Strategy** that we can use with our Configuration Management tool. This gives us three codelines, Development, Mainline and Release. Development activities are carried out on code branched from the mainline and then merged back. Releases are branched from the mainline, and client versions are cut from the release.
- We have identified the **Environments** required to put this branching strategy into place, including programmer's sandboxes, product testing environments, integration test environments, release test environments and build environments as well.

5.6 Development Partnership Programme

We have also introduced a Development Partnership with our clients. This builds upon several successful but disparate activities over the last couple of years, which have now been tied together in the Development Partnership Programme.

The first stage is to involve them in our Base Development Model, and then to leverage a number of partnership activities on top of that:

- **Assisted Development** is the process of bringing the client into our development environment, to assist us with the specification and development of the solution, ensuring that the client needs are met right from the beginning of the development process. Very much in the XP on-site customer vein.
- **Hosted Test Lab** is the hosting of a client configured test lab by us, with the pre-delivery testing carried out between us and the client test resources.
- **Remote Project Support** is the facilitation of software deliveries by our testers pre-testing in a client configured environment. This can extend the testing day, especially when working with New York and Tokyo, and also reduces expenses or makes it possible to work remotely in inhospitable areas, should we say, sell a dealing system in the Antarctic.
- **Hardware Lab** is the opportunity for the client to prove proposed or impending changes to their production environments, prior to making them, in a specially constructed test lab. Some of the client environments are quite sophisticated dealing rooms, including an exotic mix of hardware.
- **Testing Partnership** is a complete lifecycle testing partnership including the sharing of; test scenarios, conditions, and scripts, each element of which can be supported by testing consultancy.

5.7 Still To Do

But there is still quite a lot to do before we have completed the change program:

- **Roll out Unit Testing Tools**
  We will roll these out to a single team first, establish and document the process, and then take them to a wider audience.
- **CM Phase 2 – Process Change**
  This is when we implement the full branching and versioning approach. This again will be piloted first, proven, documented and then rolled out.
- **Introduce Process Workflow with CRM**
  We are currently investigating Customer Relationship Management, and it is expected that this will deliver Process Workflow which can be used to control our requirements, and manage the development lifecycle.
- **Increase coverage of the test automation suite**
  The test automation architecture is extensible and scaleable. That gives us the capability to add new functionality to the suite and also vary the load from a few to many thousands of transactions.

- **Leverage automation into other areas**
  But we will not stop at the testing team with test automation. There are other areas that will befit from using the test automation products. In the short term we will be rolling out automated set-up and configuration scripts to our build teams.
- **Review testing activities across the whole company**
  We are also undertaking a review of testing activities across the company to identify synergies and efficiencies from further tuning the testing lifecycle.

5.8 The Story So Far

The following points provide conclusions from the work to date:

- **It is hard enough just changing the testing process!**
  When you are just dealing with change in the testing team this is a challenge in its own right. Changing the development lifecycle is much harder, and by its nature involves other groups, developers, software management, analysts, senior management etc.
  You have to involve these groups, and then get the individuals within them to make the changes that you need. That is the hard part.
- **You need sponsorship and support to re-engineer the development lifecycle.**
  Without sponsorship from very senior management this level of change will not happen. You have to ensure that you have approval. We obtained board level approval to proceed with the change programme. You also need support from people who have been there and done this before, so that you don't make all of the easy mistakes.
- **Change is an inclusive process.**
  Don't forget this. You need to get everyone's buy in otherwise this level of change will fall apart very quickly. We have put a lot of effort into including people in the working groups that are making the decisions. Council widely before making change. Just telling the developers what the answer is never goes down very well.
- **Change is continual.**
  Although you have a plan to change, you have to be aware that change is going on all around you, all of the time. Our example was the desire to move to more agile development methods whilst we were underway with our change program. This is going to happen, don't fight it, rather, be prepared and work with the change.
- **And Finally, It is harder not to change.**
  We are in an industry where we can not just stick our head in the sand and ignore everything. Why would we want to continue with inefficient and unproductive processes anyway, they are holding us back? The Thin Threads model really helps here, because every change is a step forward in the right direction, in an easy and manageable way.

6 Where value was added

Following the theme for Eurostar 2002 this section outlines where value was added; to the testing team, to the development group, and to the organisation as a whole.

The following are the list of **successes**:

- **Test Process Improvement**
  Most importantly from a testing perspective the testing team was able to build upon their process improvements with clearly defined requirements to drive their test design and preparation activities.
- **On-going Change Program**
  The change program gave management a clear view of how to improve the processes across the development group.
- **Identified two-speed lifecycle**
  The two speed lifecycle did not impose an overly burdensome approach on the developers for smaller items of work which benefited from a more agile approach.
- **Tool evaluation for CM & Unit Testing**
  The developers gained a clear benefit from identifying and evaluating unit testing tools which improved their productivity and the quality of their deliverables.
- **Broadened the view of quality from QC to QA**
  The development group as a whole began to change the view that testing was just a quality control activity towards a quality assurance mentality.

And of course, not everything goes smoothly so here is the shorter list of failures to add value:

- **Point solutions only**
  One of the commercial realities in small companies is that there is not always the money to start all of the projects straight away, and in that respect the change program did suffer with the focus clearly being placed on point solutions rather than an integrated change across the board.
- **No quick wins**
  A large and complex program which involves re-engineering the development lifecycle, implementing configuration management, etc, does not always generate quick wins and has high start up costs for tools and training. This definitely counted against the program.
- **Pick & Mix approach by development**
  Senior management took the view that the change program roadmap was a pick and mix option, choosing only to implement items which would not cause any short term impact upon the development workload.

7 <u>References</u>

The following references were used in compiling this paper and the associated presentation:

British Computer Society Specialist Interest Group in Software Testing Standards Working Party meetings 2000-2002 inc.

KOOMEN, TIM and POL, MARTIN (1999) Test Process Improvement.

HOLCOMBE, M (2001) Extreme Programming: What it is, does it work and what are its implications for the testing profession? British Computer Society Specialist Interest Group in Software Testing, July 24.

POL, MARTIN (2002) How to improve your testing process? British Computer Society Specialist Interest Group in Software Testing, February 12.

BECK, KENT (2000) extreme programming explained.

FOWLER, MARTIN (2001) Keynote address XP Day, Agile Software Development and Extreme Programming, December 15.