Adventures in Session-Based Testing

Authors James Lyndsay, Workroom Productions Ltd, Niel van Eeden, Jobpartners Ltd.

Biographies James Lyndsay is an independent test consultant with ten years experience. Specialising in test strategy, he has worked in a range of businesses from banking and telecoms to the web, and pays keen attention to the way that his clients' focus is shifting away from functional testing.

Niel vanEeden has a background in mechanical engineering, IT hardware, software and retail. Since relocating to the UK from his native South Africa, he has worked in customer facing roles and in software quality, and has been involved in testing at JobPartners since May 2000. In January 2001 he became the test manager, and is currently responsible for product quality.

Abstract This paper describes the way that a UK company controlled and improved ad-hoc testing, and was able to use the knowledge gained as a basis for ongoing, product sustained improvement. It details the session-based methods initially proposed, and notes problems, solutions and improvements found in their implementation. It also covers the ways that the improved test results helped put the case for change throughout development, and ways in which the team has since built on the initial processes to arrive at a better testing overall.

Session-based testing can be used to introduce measurement and control to an immature test process, and can form a foundation for significant improvements in productivity and error detection.

Keywords Ad-hoc testing, Session-based testing, Functional testing

Session-based Testing

Session-based testing is a technique for managing and controlling unscripted tests. It is not a test generation strategy, and while it sets a framework around unscripted testing, it is not a systematic approach whose goal is precise control and scope. Rather, it is a technique that builds on the strengths of unscripted testing - speed, flexibility and range - and by allowing it to be controlled, enables it to become a powerful part of an overall test strategy.

At the heart of the technique is the idea of effective limits. A Test Session has a well-defined start and end time, limiting its duration. During a Test Session, a tester engages in a directed exploration of a limited part of the thing being tested - it should be obvious to the tester that an action or test is inside or outside these limits. Within these limits, moment-to-moment activities are not controlled, but left to the tester's judgement. The tester records his or her activity - and includes whatever other information seems relevant; the reactions of the system, data used, conditions, diagnosis or ideas.

Session-based testing mirrors the activities of experienced testers, but is not the subject of a great many papers or books. This paper describes one situation in which session-based testing was successfully implemented.

Context

The product to be tested was an application delivered over the internet, and had been commercially live for just under a year. The application had a few hundred active users at a few dozen firms, and dealt with a large amount of incoming data submitted by many thousands of internet users,.

This application had been developed in-house by a medium size team (30-40 people total). The team continued to develop the application, and released a new version of the application every two weeks or so. Work was driven by a semi-formal change request process.

Although customers were satisfied with the overall service, there was a perception within the company that the quality of the product had to improve. Something in the development process (requirements, analysis, design, coding, testing, infrastructure, release) had to change - and unsurprisingly, attention initially concentrated on the test process.

The existing test process was immature, and the five-member team had little experience. None of the team had experience of a well-run test process. The precise nature of the problems cannot be detailed in this paper, but the process exhibited the following common characteristics.

- Reactive - and therefore uncontrolled, and not necessarily focussed on important areas
- Could miss important bugs which had an immediate effect on customers
- Could not produce reliable information about the readiness of a release, and was not trusted.

The test team were active users of the mature bug tracker Bugzilla. This tool was central to the team’s processes, and drove much of the fix/retest work in the coding and testing teams. They had been using the tool for the life of the product, and had a well-established bug list.

To help initiate the changes, the company bought in experience, and engaged one of the authors of this paper (James Lyndsay) for forty days spread over ten weeks.

Constraints

With a mandate for change, management were supportive of changes within the test team and their test processes. However, the changes made had to stay within existing budget and resource. The test effort needed rapid improvement, yet at the same time, the coders were to increase the rate of introduction of new features, and many known bugs would be fixed for retest.

Existing testing found good bugs, but in a bad way. Some of the most significant bugs would only be found by ad-hoc tests - particularly those caused by data problems and often characterised by intermittent symptoms - and the team were reluctant to move away from a proven approach.

The team were already stretched. Any time spent on training or setting up new procedures would have to be saved elsewhere. However, as the existing process was inefficient, time savings were not hard to find.

Basic strategy for test improvement

We needed a simple process which would improve on the existing ad-hoc methods. To allow us to do this, we had to introduce some element of measurement, so that we could see which parts were working, and which parts were not. We also needed to introduce control, so that we could define our tasks, record our actions, and so repeat or improve them.

Canter/Derr had done work at e-greetings.com which inspired the team. Their paper (*A Case Study in Extreme Quality Assurance*, referenced below) helped the team to believe that they were not alone, and that similar problems and pressures had been overcome in other organisations. However, Canter/Derr's approach introduced change well outside the test team, and while desirable, could not be implemented under the existing mandate.

James Bach's exploratory test methods (referenced below) meshed well with James Lyndsay's existing 'Empirical Test' techniques, and Jonathan Bach's practical implementation of session-based test methods (described in the paper *Session-Based Test Management*) seemed to offer a useful and practical starting point. Bearing in mind the need for a simple, lightweight process that would form the basis for ongoing improvement, we set out to do the following things:

- Control the scope of testing
- Control the work
- Measure risk and coverage

Session-based methods

Session-based testing parallels the way that many experienced testers approach ad-hoc testing. While not a new technique, it has not been formalised - and there are no hard and fast rules to its execution. However, session-based testing is characterised by (at least) the following:

- A test session is a unit of time, generally a couple of hours long. It is uninterrupted, as far as possible, and its limits are well-defined.
- During a test session, testers test something specific. They may test a feature, a characteristic, a business scenario - they may hunt bugs or introduce failures. These limits are less well-defined, but they are defined *before* the start of the test.

By introducing these limits, session-based testing seeks to focus tester attention, allowing control, increasing the reliability of metrics and the repeatability of tests, and limiting the cost of poor exploration.

The methods below allowed the team to:

- Control the scope of testing
- Control their work
- Assess coverage
- Assess risk and set priority

Controlling the scope - introducing Test Points

The team had no existing test list, and the project as a whole did not have uniquely-identified requirements. Each release introduced a wide range of new tests, and although test scope was driven in part by bug fixes, there was no list of new tests or record of tests done.

We needed some sort of a list of tests, to enable us to:

- select tests and so drive work
- consciously omit certain tests
- make easy and repeatable assessments of the state of testing
- avoid duplicates
- allow forgetting without loss
- simplify communication within the team, and extend communication outside the team
- generate reliable statistics

The items in this list might be tests, but the concept of a test in unscripted testing is very different from a scripted test. We tried to avoid some of the characteristics of unscripted testing by carefully choosing the things we would control. The items on the list:

- Would not be single tests, but aspects of the application that needed to be explored. A tester would typically conduct many tests in their exploration.
- Needed to be independent units of work - not steps along a path. A tester could start or finish with any item.
- Needed to be based on a wide range of sources. These included change requests, functional requirements, development information, release notes , regression test requirements - and on conversations over lunch, inference from unspoken topics, eavesdropping, wiretapping and covert midnight operations.

To avoid confusion with 'tests', we call these items Test Points. Our test points have the following broad characteristics:

- A Test Point is a unit of work and typically takes between 20 minutes and 4 hours. This estimate of duration is first made at the point when the Test Point is defined, and can be used as a simple metric for the *cost* of the test. It is refined during testing.
- Each Test Point has a simple *risk* assessment. This assessment is also done as part of the process of defining the Test Point. If a Test Point has a range of risks, it is split.
- Test Points are kept from release to release. Some Test Points may only be explored rarely, some become part of a set of regression Test Points, some crop up each release but their exploration changes as the functionality changes.
- Every piece of test work has its associated Test Point - including test work from more formal methods and work for non-functional testing.

Example Test Points:

- Is a field for 'Salary' offered as an optional input at all appropriate points?
- Examine User Access Control, using usertype *xxx* and usergroup *yyy*
- Does the 'Forgotten my password' option ever fail to send an email?
- Check navigation in 'Options' part of application, paying particular attention to 'back' button functionality within application, and within browser.
- Check button text within 'Options' part of application for each language offered

The list of Test Points is dynamic - additions are made frequently, based on bugs found, new understanding, surprise functionality and fixes delivered. We currently hold the list in a spreadsheet that can be accessed by the team at all times, but the same job might be done as well, or better, by a database accessible over the internet. The list can only be changed directly by the test team, but can be (and is) accessed by many others. Regular users outside the immediate test team include the Development project manager, Professional Services and senior management.

Each Test Point is associated with the following information:

Test point ID	Unique ID. We found it useful to include the release number in this unique ID. Test Points related to bugs also had the bug ID as part of their unique ID.
Title/Description	Enough to set the boundaries of the test
Risk	Simple, repeatable risk assessment. See below for details
Estimated time to complete	If the estimated time to complete is >4 hours, it may be an indication that the TP should be split into two simpler parts. It's hard to do useful exploration in less than 15 minutes, so a very short time might indicate that something needs to be re-assessed. This estimate is updated after test execution to be time spent ÷ % complete Also called 'Cost'
Time spent so far (this release)	Updated as test sessions are completed. Reset after release - so it records the time spent to far this release.
% necessary testing completed	Basic estimate. Will be at 100% if testers feel that the TP has had enough attention to be passed to the customer. May start at 100% if TP has no need to be tested. Will generally increase with work done, but may go down if the TP looks like it needs more testing.
Time remaining	Calculated as Estimated time to complete * (1 - %complete)
Tester	Named Tester, responsible for this Test Point
Documentation	Cross-references to relevant documentation

We needed to have the best assessment of risk and cost, and felt that this would be made most reliably by those closest to the Test Points - so individual testers were made responsible for Test Points. This had the added effect of motivating the team and neatly defining their test tasks. Responsibilities include:

- identifying the documentation – and raising notice of its absence
- ad-hoc testing around these Test Points
- raising bugs found
- filling in the Test Session Report
- updating Test Point information – risk, time spent, necessary testing completed
- adding new Test Points for the release if they feel it is necessary
- talking about the Test Points at group meetings

Controlling the work with Test Sessions

While a Test Point might be described as a unit of work, a Test Session is a unit of time. A Test Point may be repeated - a Test Session is planned, happens, and is recorded. Each is unique. By setting the scope of individual tasks, controlling the time taken to do them, and requiring deliverables on completion, we controlled the work, and were able to dynamically adjust the plan.

A Test Session is an uninterrupted period, generally half a day or less. During a Test Session, testers will investigate one or more Test Points - with the minimum size of a Test Point at around 20 minutes, most Test Sessions look at no more than four. The choice of the Test Points to include in a session can be made on a number of criteria, and for planning, works well when related Test Points are chosen together. However, in action, it tends to be a very dynamic process, reflecting the team's need to react to the fast-changing priorities that are characteristic of a rapid-release environment. A plan that cannot adapt to fit circumstance is worse than useless in a changing environment.

Typically, the Test Manager controls the initial choice of Test Points, and also acts as the driver for change when necessary. His/her decision is based on:

- Availability of software
- Availability of test resources
- Time needed to test vs. time available
- Tests done so far, and their coverage of functional areas and risk
- Existing Plan

Session Timeline and Reporting

Before starting a Test Session, Testers print off a blank test session report and fill in the administrative details - including the Test Points they plan to explore. Test Session Reports have space for the following information:

Test Session ID	Unique ID. It was helpful to include information about the release, and about the tester doing the test session - although note that including the release ID could both cause and resolve confusion when testing (particularly fixes) in a release other than the one being used by the rest of the team.
Title/Description	Enough to set the boundaries of the session
Contents	Test Points covered in the session (often 1, rarely more than 4)
Administrative details	Tester name, release, date + time etc.
Notes	To be filled in while testing, although system conditions, available data etc. is often noted before the start of the session.

While exploring those Test Points, the testers use the Test Session report to record data, impressions, problems, queries, possible bugs, bugs raised, diagnosis and so on. The reports aren't formal, or neat, but they are good records - and got better as the testers became more experienced.

A Test Session is a timed activity. At an appropriate point around the end of the allocated time, the session ends. Testers are not encouraged to spend more time than planned on any one Test Session - although the Test Points may be returned to in a later session.

The deliverables at the end of the Test Session are:

- completed session report filed appropriately
- updated list of Test Points

It is important to remember that although the Test Session may be finished, testing may not be over. Although ad-hoc testing can be controlled by imposing an end-time, problems found during testing can result in a greatly increased - or decreased - estimate of the time necessary for adequate testing. Dealing with this was an important factor in deciding what to measure.

Review of Test Session Reports

Although simply writing a Test Session Report helps the testers (see below), recording the events allows Test Sessions to be reviewed after the event. This helps different people in a wide range of ways.

- Helps testers and the Test Manager when setting the severity of a bug, looking for duplicates etc. It is particularly helpful when the severity is queried
- Helps decide how to approach testing if the time needed is more than the time taken so far.
- Helps the Test Manager and individual testers to control and improve the quality of testing. The Test Session Report is a useful coaching tool; the coach does not have to sit with the tester for the duration of the test, and more than one session can be reviewed and compared. The Test Manager also gets a good idea of the approach to testing of each tester, and can re-direct as appropriate.
- Helps the test team to look back on a session, to be reminded of their actions and results, to examine the data used in the light of new information etc. Reviews also allow testers to re-interpret their conclusions, or to use multiple session reports for diagnosis or examples.
- Helps testers to share information - two testers might approach the same Test Point in different, complementary ways, or use a Test Session Report when handing over tasks.
- Helps coders and designers get over the 'if I haven't seen it, it isn't a bug' problem. Illustrating the problem by showing the data used can be very helpful, and noting times helps to match problems with known system events - network failure, batch job etc.
- Having a tangible and discussible record of test work available for review helps the business have confidence in the test process, and the reviews themselves help the testers and the business understand each others priorities and desired approaches.

Assessing test coverage

Coverage - a brief explanation

Coverage is a measurement of what has been done as a proportion of what could be done, and is an important test metric. It plays a key role when assessing whether the product has had enough testing to give the business confidence that it can be released. When broken down by area, it can indicate those aspects of the product that have had enough, or too little testing, and so becomes a vital input in adjusting the plan as circumstances change.

There are many ways of measuring coverage - Cem Kaner lists 101 coverage metrics in his paper Software Negligence and Testing Coverage, some serious, some not so serious. Although they are each measures of 'testedness', they do not all measure the same thing - so 'good' coverage by one metric may be inadequate when measured by another (i.e. Statement coverage vs. Service Level Requirements coverage). It also can be hard to measure some coverage metrics without instrumentation set up prior to testing (i.e. branch coverage), and others can be impossible to measure in some projects (i.e. requirements coverage in a project without formal requirements). Indeed, while comprehensive testing results in good coverage measured by most methods, testing driven by a single formal technique can result in excellent coverage when measured by one method, and poor coverage when measured by another.

Formal methods of coverage measurement do not work well with unscripted testing, and can introduce complexities to test execution that work against many of unscripted testing's better qualities. We introduced a measurement of coverage that was simple to assess yet gave a good indication of the testing that had been done compared to what needed to be done, the readiness of the system for live operation, and the parts of the system that needed more testing.

Our coverage metric

We based our coverage metric on a subjective assessment of 'testedness'. By using Test Sessions to focus and control the work and take many small-grained assessments, we hoped to be able to make an objective measurement from the combination of subjective estimates.

At the end of a test session, the testers recorded two figures;

- the amount of time they had spent testing each Test Point.
- an estimate for how ‘tested’ the test point was, as a percentage.

As these figures were recorded, other figures used in planning future work were calculated (new estimated test cost, time remaining, overall coverage). Calculating these figures on the fly gave the testers immediate feedback and a more concrete perspective on their estimate, helping them to make their estimates more consistent.

Example: *A Test Point was estimated at 3 hours. The tester completed 3 hours exploration, but felt she had not yet tested all the things she wanted to test, and estimated she was 75% done.*

On recording these figures, two figures were calculated automatically. The Test Cost was raised to (3÷75%=) 4 hours. The time remaining was set to the 1 hour that remained.

	Test Point	Cost	Completed	% complete	Cost/time
Before	assess email functionality triggered by …	3 hours	0	0%	3 hours
After	assess email functionality triggered by …	4 hours *calc*	**3** *input*	**75%** *input*	1 hour *calc*

We hoped that the tester’s expertise would enable them to make a fair judgement – but recognising that the testers were not yet expert, we encouraged the testers to spend time looking back over past estimates and discussing their accuracy – and, if necessary to re-assess their current estimates. The team found this useful, and were also helped by the immediate feedback from their estimates into figures used for planning.

Special cases -

- Test Points that needed no testing were at 100% coverage from the start. This was achieved by setting their test cost to 0 and including a special condition in the number crunching functions,

as keeping the cost and setting work done to match it introduced confusion. Note that such Test Points are excluded from some summaries.

- Tests where the tester had done three hours - but kept the estimate at two - were not listed as 150% done, but were fixed at 100%. However, they were highlighted - to highlight input mistakes, and to allow the Test Manager to review the test with the tester and avoid the issue next time.
- A 1-hour test that the tester estimated was 10% complete needed 9 more hours. Some tests genuinely needed this much more testing, but others could be more accurately re-estimated, or given special attention by experienced team members or the business to see if the testing could be made more efficient. Giving feedback to the testers by exposing the planning figures helped avoid this problem.
- At the start of testing for each release, the amount of testing completed for each Test Point was reset to zero, and estimated cost and risk was re-assessed for existing Test Points.

Combining estimates for more accurate overall figure

Although each individual estimate was rough and subjective, their worth improved when they were combined. Note that a less blunt combination, split by risk, would be used for most decision making. Estimates were combined as follows:

- Overall test time remaining, derived from the sum of (estimated time - completed time). This was useful throughout testing - it gave an estimate of the effort needed at the start of the process, and tracking it in real time allowed the Test Manager to see how well the team was staying on target. It is worth noting that there was always more testing to do – we never reduced this figure to zero.
- Coverage for the whole release, derived from the (sum of time completed)/(sum of estimated time to complete). This gave a one-figure summary of the progress of testing. Just as the % complete of individual test points could go down as well as up, so could the coverage. This was entirely appropriate - and was usually the result of the addition of new test points from bugs, or as a result of underestimation of test cost / complexity.

Conclusion

Testers are notoriously bad at ad-hoc estimates of how much testing is needed. We aimed to improve this ability – and, in part, we achieved this by placing the skill at the centre of the planning process.

These metrics assessed not the state of the system, but the state of testing. While necessarily subjective, the metrics turned out to be repeatable - different testers came up with broadly the same estimates of completion, particularly as they worked together, and grew more experienced in their methods, in testing, and in the system. Because the system was assessed in small grains, individual errors in estimation at a Test Point level were small compared to the overall figures. The team updated their figures regularly and often, and the metrics became an important feedback to the team, helping them perceive a common goal and giving good feedback into their process of ongoing improvement.

Management understood that the metrics gave a reliable indication of how the testers felt about how well they had tested the system. This, in combination with the number, type and severity of problems found and fixed, was soon an important part of the go-live decision. Rather than wait until the end of testing to find out how good the system was, the decision could be assessed earlier, allowing warning of problems and re-prioritisation of effort. The ‘coverage’ figure was both useful and effective.

Risk and Prioritisation

The test team needed to be reactive. Their reactions needed to be fast - but could not be allowed to be uncontrolled. They needed a means of assessing new circumstances against existing tasks quickly and accurately. In making decisions about importance and urgency, it was good to have information about risk and cost.

Risk is a combination of cost of failure and likelihood of failure. To allow assessments would be consistent and repeatable, we needed a simple method of assessing risk. We started with possibly the simplest:

		Likelihood of failure	
	Risk	High	Low
Cost of failure	High	3	2
	Low	2	1

Each Test Point was assessed for risk. Work was assigned by risk, and metrics were split by risk. We found that the risk associated with a Test Point was easily communicated to the business, and broadly matched their expectations.

Note that test work did not concentrate simply on the highest risk test points. It was important to spread the test effort in case the risk assessment was wrong, and we typically spread the test effort so that all major functionality had some testing. However, at the end of testing, high risk Test Points generally had better coverage than low and medium risk Test Points.

In some releases, coverage of high-risk test points never matched the coverage of low- and medium-risk test points. This was due to two factors

- high-risk elements released to testers / fixed close to live release deadline
- the more testing that was done in high risk areas, the more the testers felt they needed to dig deeper

We found that the proportions of Test Points over time matched the following profile:

Risk	% TP by number	% time required	% time spent
3 (highest)	15%	20%	25%
2	35%	40%	40%
1	50%	40%	35%

This profile indicates that individual high risk test points generally required more time than low-risk ones. The difference between %time required and %time spent indicates that when the team were running out of time, high-risk test points were given priority over low-risk test points.

Note that these figures include a substantial proportion of regression-test-related Test Points, which are generally judged low- and medium- risk. This lowers the *proportion* of high-risk tests. If these figures were for a single release concentrating more on new functionality and fixes than regression testing, it would have a greater proportion of high-risk tests.

Building on the Process

The new methods were not adopted immediately, but good initial results encouraged the team to persevere. The methods have been in place since June 2001, and have been supplemented with a number of improvements and refinements.

Process and Techniques

- Some Test Points now include scripted and automated elements if appropriate and effective.
- Test Points are grouped not only by risk, but also by family. The four families are:
  - Test Points for new functionality
  - Test Points for fixes
  - Test Points for regression testing
  - Test Points for QA investigation
- The Test Team has a process of ongoing learning:
  - A ten-minute daily meeting keeps the team focussed and together, and highlights common issues
  - A two-hour training session, each Friday afternoon, helps the team take a longer-term look at the application, at test techniques, tools and at process improvement. It allows them to share their experience, and encourages them to think of testing as a skilled job.
  - The figures and feedback have helped the team to improve their estimating skills, and have encouraged their planning skills
  - The practice of recording their test sessions allowed review and improved their ability to test without scripts. Sessions were reviewed by peers, by senior testers, and by the testers themselves after the fact.
  - The responsibility for individual test points encouraged ownership and interest, improving test analysis and planning skills
  - Communication improvements driven in part by test sessions encouraged skill sharing and greater interest in the general process of testing and its literature

Metrics

- Refined figures improve the focus for individual testers and Test Point families
- Automated metrics provide a 'Test Dashboard', giving instant feedback of overall summary and planning figures as test are completed
- Improved metrics allow more complete views and more reliable assessments
  - Identification of bugs found in live has allowed the team to start measuring test effectiveness
  - Measurements of estimation accuracy help improve the estimation skills of the team

Early involvement

- The Test Team is now involved earlier in the process, and is able to discuss design and assess code before delivery to test
  - Testers are involved in document review – and now find problems before coding
  - Testers spend a short period testing code with a coder, at the coder's machine, after unit testing but before promotion to the generally available code. Not only do the testers increase their familiarity with the deliverable (and the coder), but they also spot simple problems that have not been apparent from the unit tests.

Changes outside the Test Team

- Improved documentation references allow testers to link each test point to each identified document and track dates for its completion and availability. Documentation availability has made work more intensive, but more focussed, and the team is able to be more productive

- Better documentation has improved the linkage between tests and business requirements
- Visible improvement in the Test Team encouraged process improvement elsewhere, particularly in the generation of inception, design and implementation documentation, and in the processing of urgent requirements for live fixes.

Results

Perhaps the most significant result is that testing is now seen as a vital aspect of the development process, rather than incompetent, obscure, and a hindrance to productivity.

'Tangible results' listed below are those characterised by a deliverable or directly measurable quantity. 'Intangible results' are those characterised by a change in perception.

Immediate, Tangible results

- The team produced a useful coverage metric from the first pass through testing, and showed that riskier parts of the system had received more attention. This has subsequently become a central metric.
- The Test Manager was able to review testers' work off-line i.e. without being with the tester while the work was done.
- Test Session reports were a useful record of ad-hoc test activities, where previously there had been nothing but a bug report.
- Because few measurements had been taken before the introduction of these methods, it was hard to get real results in the early stages. However, the rate at which significant bugs were found stayed the same on the introduction of these methods, and increased for the next five months – although this reflects the increasing complexity and size of the code (as so the number of bugs to be found) as well as process improvements.

Longer term Tangible results

- The product is more stable and has fewer outstanding bugs
- In the last few months, the rate at which significant bugs are found has fallen, although new functionality is still being introduced as fast as ever. This reduction is thought to be due to the increasing quality of the code, rather than test failure. The test team’s skills and procedures have been instrumental in helping the designers and coders achieve this improvement.
- The test team’s metrics are used as a basis for improvement by non-test teams.
- Problems outside the test team were no longer obscured by test team problems, and could be identified and addressed. This applied particularly to documentation, which was refined to fit a useful purpose rather than simply generated as part of a deliverable, and to the way that live problems were handled.

Immediate, Intangible results

- The test team felt in control of their work. They could see the size of it, see how much they had done, and what was left. They could decide what to do next, and back up those decisions.
- The Test Manager felt more confident in controlling and planning testing.

Longer term Intangible results

- The coders felt that problem logging and diagnosis had improved
- Visibility of test process and progress allowed other teams to trust the Test Team's information, and the communication that the trust enabled resulted in a ‘virtuous circle’.
- The introduction of more formal, scripted testing was easier as the test sessions helped the testers to think more rigorously, and to work in a systematic and analytical way.
- The test team take a much greater interest in their jobs, and morale has improved. The team generate three or more good ideas a week, of which at least one is implemented.

Lessons Learned

The team needed a process that enabled learning and encouraged improvement. We recognised that while we might not start out well, the right attitude and the right tools would allow us to develop an effective and efficient process.

Three overall factors

While we learnt a number of useful lessons, three key factors stood out. These factors underlay many key parts of the approach - without any one of them, the approach would have failed

- Communication. The methods above gave us the tools to communicate within and outside the team. By improving communication, we felt that we reduced the number of misunderstandings. Communication also helped to increase trust, which both improved personal relations, but also helped facilitate solutions.
- Empowerment. Testers were individually responsible for Test Points. They were encouraged to measure their own progress and their estimates were trusted. Morale improved, and the test team was seen as an interesting and valuable place to work.
- Openness. The list of Test Points, the work done and the work needed were available to the coders and designers at all times. Although initially attracting little interest, the fact that the information was always available, and always up-to-date, encouraged the other teams to work with the test team, take an interest in their activities, and trust their work.

Cost estimation

Test cost - in terms of the time a test would take - was a vital metric. By comparing the actual cost with the initial estimation, we hoped to improve estimation skills.

Within the first couple of cycles, it became obvious that the whole team were not only underestimating the cost of risky tests, but their estimates got worse for longer tests. This was bought to the team's attention, and estimates improved. Analysis of a recent release indicated that 5% of tests done would have required more than twice the estimate to be fully tested, and that estimates were within 35% of the required time for 70% of the tests. Over the whole release, the time the testers felt was needed to fully test the release was 25% more than their original estimate – but note that this straight average is deceptive – tests that need less time than estimated cancel out those which need more.

We believe that the testers are accurately estimating the time needed to explore a Test Point to an acceptable level. This is supported by the improvement in test effectiveness (bugs found in live / all bugs found). This is an important skill, and allows the Test Manager to plan and react with confidence. It also allows the rest of the business to trust the testers estimates.

Test Points - analysis

The analysis needed to define a list of Test Points was not trivial, but the process of generating Test Points gave form and repeatability to a necessary analytical task that was not otherwise addressed. Although this analysis was unfamiliar, it was easy for the test team to see when they had finished - they had a definition, a cost and a risk. As releases and test cycles passed, the team got better and faster at doing the analysis.

Test Points are likely to overlap - particularly when defined by someone who is not familiar with the list. The Test Manager plays an important role in identifying duplicates and overlaps, but his/her job is made easier because the list is public. Resolution is made simpler because each Test Point has a Tester assigned.

The task of defining Test Points has been made easier by improved documentation and tester involvement in design and implementation meetings. The team finds that not only does the increased familiarity with the requirements help, but that the extra time to think improves the scope of their analysis.

Test Points - writing descriptions

One of the difficult parts of writing Test Points is to define a non-trivial exploratory area that is well-defined enough for the testers to know what is in the area, and what is not. Consideration of three aspects helped:

- time limit - areas that might take a couple of hours to explore were easier to define than larger areas.
- risk - if an area of exploration had risky parts, and not-so-risky parts, it was probably two Test Points.
- wide range of different approaches that could be taken - one Test Point might examine the functionality of 'Back' buttons and return navigation throughout the application, while another might look at ways that email-sending functionality could be broken.

Test Sessions

Testers may find it productive to collaborate on a test session, particularly when the area is unfamiliar. Some test sessions, for functionality hidden from the users, may be performed in collaboration with the coders. Collaboration with coders is also used in short test sessions (called ‘splash testing’ on-site) that are performed immediately before the coders incorporate new functionality into the main body of code.

Test Session Reports

We found that the act of writing stuff down encouraged better testing, as testers could refer back to what they had done, and leave distractions for later without losing track of them entirely. They could draw diagrams, annotate previous notes and use colours and sticky labels – and, under pressure, most testers found it faster to write than to type (this may because the testers use the same PC to test as to run the word-processor). Testers using paper documentation did not have to worry about a PC failure causing the loss of their session log. We also found that when the sessions were reviewed, a hand-written log was a better visual mnemonic than a typed or on-line document.

It is worth noting that each member of the team has a different style of testing, and each produces a slightly different style of test session. One of the team feels that the advantage of hand-written documents are outweighed by the ability to use copy/paste – particularly given the legibility of his handwriting – and prefers to use a word-processor and other PC tools to record his Test Session report.

Testers got better at writing session reports - partly as a result of reviews, and partly as they started to use the reports as a tool in themselves

Maps

We found that a map of the navigation of the product helped the testers with aspects of the system that manifested in many places, acting both as a breadcrumb trail and as a checklist. It also helped them plan their testing and estimate completion more accurately. However, the map cannot be constructed automatically and the team has had problems with obsolescence.

Documents needed

An important change in the development process was the introduction, enforcement and tracking of standardised documentation. These documents helped the testers explore areas more effectively, and the tracking helped them plan their activities to match the design and coding teams schedules and events.

- Inception Document: contains the original idea. Describes the way the new feature needs to work, contains the requirements and the design logic.
- Design Document: Analysis of changes needed to database, classes, modules and pages
- Implementation Document: Details changes actually made. Signed off on delivery of code to test.

Rapid reactions and real-time results

Once the tracking spreadsheet had been set up to include a real-time test dashboard, the Test Manager always had an up-to-date picture of the tests that had been done, the tests yet to be done, current issues, coverage and risk. This knowledge allowed the team to react more quickly to changing circumstance, without losing track of the overall aims of testing. The improved response had a direct effect on the way that other teams and staff interacted with the test team, and increased trust and communication.

Fixes, retests and regression tests

The team quickly adopted test sessions to drive and record retests and regression tests. Test sessions allowed a faster response to the arrival of a fix, and served as effective proof that the fix had been well implemented and tested. The team found that looking at the session for the test when the problem had been found helped plan the retest.

Test Points are now classified into four families (see "Building on the Process" above), one being 'Regression Test Points'. This important improvement has resulted in the development of a comprehensive regression test set. A selection of Test Points for 'new functionality' is added to the set of Regression Test Points at the end of each release, keeping the tests current. The team can isolate coverage figures by family, allowing ongoing assessment of the depth of regression test coverage for the release.

Outside the team

Other parts of the organisation can understand and read test points – and the figures summarising test progress are available at all times. The business may add to the test points, but in practice will always ask a tester to add any that may be required. This allows control, and directly assigns ownership of the test point to an individual.

We found that having improved the test process and the visibility of its results, other teams started to change their processes, as thin spots were revealed. Changes in coding practice and in the preparation of the design have been initiated partly because testing could reveal and measure the points where existing practices were not working well. Embarrassment and peer pressure can be an important motivating factor in the improvement of code quality!

Live bugs

With session-based testing, we were able to get real value from analysis of live bugs. We could look back over sessions for the current release or previous releases, and could analyse the tests done to discover how the bug had been missed. This approach, impossible with poorly-recorded ad-hoc testing, drove a multitude of small process improvements.

Mistakes and problems

We drove the testing from a single, complex spreadsheet. While this allowed good flexibility and quick improvements, it caused a number of problems:

- Corruption and data loss: The spreadsheet was shared – and sharing did not work perfectly. Summary test metrics were helpful in identifying corruption, and a few 'sanity checks' were built in. The spreadsheet was backed up regularly. These problems have become less frequent as the team have become more familiar with the spreadsheet application.
- One line per test point: A spreadsheet is not database. In particular, it does not allow a simple method of recording many actions to one item, as each item is recorded on a single row. In this case, the restriction made it hard to input and extract good information for test points which were performed more than once in a release (inclusion in more than one session, re-delivery of software after a fix, poor approach the first time etc.). It also caused problems when a test point was performed by two testers working together, or when adding multiple, dated comments. There was no easy solution to this, but the numbers involved did not unacceptably compromise the accuracy of the metrics.

There was no explicit link between Tester, Test Session and Test Point – each Test Point had a Tester, and each Test session had Tester and Test Point, but there was no linkage to allow the an extraction of all the Test Sessions that had involved an particular Test Point. While this seemed important in planning, in practice the close ownership of Test Points by individual Testers meant that the information was easy to reach.

Close ownership of test points meant that testers were unfamiliar with some aspects of the application – which could lead to poorer testing if one of the testers was unavailable. Once a family of regression tests was developed, ownership of the regression tests was rotated each release to give each tester exposure to the full application.

Including the release in the ID of the test point and the test session made good sense during the first few releases. However, it could become confusing when re-doing test points that had been generated

for previous releases, or test points which were to retest an area following a fix. We now include the release ID in some, but not all of the Test Point IDs and Test Session IDs.

Naturally, Test Sessions were rarely uninterrupted. We also found it difficult to separate out time spent setting up / clearing down the test, time spent doing the test, and time spent logging bugs. Some activities that might be considered part of exploratory testing - such as talking to the coders and designers about the system and its problems - were not generally part of the test session. Work is in progress to address this.

Testers dealt with large amounts of information – and the volume of the information means that sessions can contain errors and omissions. Ideally, the testers would have liked to copy/paste directly into their handwritten session logs – but without a budget to create an impossible tool, they printed out information and stapled the printout to the session. This needed a nearby printer to avoid upsetting flow of work.

The simple risk assessment worked well in the initial stages. However, the business and the testers soon demanded a more refined scale, although no method has been decided which allows consist assessment by different people. A priority field is currently being used in conjunction with the risk assessment to plan testing.

The team are very happy with session-based testing, but this has led to some resistance to systematic methods and automated tools. The team are, however, finding success in fitting systematic methods, automated tools and scripted test cases into their familiar system of Test Points and Test sessions.

Failures

It seems obvious to perform related Test Points in the same session, and the testers found that this was an efficient approach. However, it may bias the testers assessment of the length of time a test needs – two related Test Points performed together will need less time than if performed separately.

Any testing driven by a single coverage metric is flawed - and the methods described above are indeed driven by a single coverage metric. Splitting the Test Points by risk and into families helps with this, but it would be good to see (for instance) requirements coverage being assessed simultaneously.

Each TP is a slice through the system – many overlap, and this can be seen as inefficient testing. However, inefficiencies tend to be concentrated on commonly-used parts of the application and would still exist in other approaches.

Test sessions, as hand-written records, cannot be parsed electronically, and any statistics gathered are based on information logged by the testers in addition to the test sessions.

We are disappointed that the our test effectiveness metrics are historically supported by only anecdotes, as the data has been lost. We hope to be able to extract it in the near future.

Conclusion

Session-based test techniques worked well on the occasion described in this paper; while staying within budget and using existing resources, they allowed ad-hoc, unscripted testing to be controlled, refined and to add real value. They may be less effective in a more sophisticated environment, and they are not appropriate in environments that require systematic and complete approaches to test definition. However, by bringing control to unscripted testing, session-based techniques are a useful addition to the test arsenal.

In implementing this approach, we used a number of project-specific measures - as described in this paper, the methods may not fit other projects. However, they shared the following principles:

- Simple measures are the best
- Favour effective communication over knee-jerk documentation
- Unobtrusive, immediate metrics allow real-time control

Our experience has shown that, when given appropriate feedback, testers can learn to improve both the effectiveness of their unscripted testing, and the accuracy of their estimates. Central to this process is a repeatable and trusted coverage metric which allows many *subjective* assessments to be gathered into an *objective* view of the degree to which the product has been tested. Session-based testing allows the subjective assessment to be controlled so that it can be drawn together in this way.

Appendix 1: References

Session-Based Test Management
Jonathan Bach
http://www.satisfice.com/articles/sbtm.pdf
also STQE magazine V2, I6 - 11/2000 and STARWest 2000 conference notes.
also see James Bach's wide range of articles on context-based testing and other practical techniques at http://www.satisfice.com

A Case Study in Extreme Quality Assurance (XQA)
Authors: Jim Canter/Liz Derr
http://www.stickyminds.com/docs_index/XDD2561filelistfilename2.zip

Near Zero Undiscovered Defects and Shorter Time-to-Market!
Authors: Jim Canter/Liz Derr
http://www.stickyminds.com/docs_index/XUS202021file1.doc

Software Negligence and Testing Coverage
Cem Kaner
http://www.kaner.com/coverage.htm for 101 coverage metrics

A Guerrilla Guide to Empirical Testing
James Lyndsay
http://www.workroom-productions.com/papers.html (may not be available for Summer 2003)

Appendix 2: Acknowledgements

The authors would like to thank the designers, coders and testers involved in the project for putting our ideas into practice and for supporting us with so many fine ideas. We would also like to thank the reviewers for their patience and suggestions.

Appendix 3: History

Version	Description
1.0	Original. Winner of 'Best Paper' at STARWest 2002 and EuroSTAR
1.1	Cosmetic changes, added Appendix 3: 'History'
1.2	Appendix 4: Addendum describing later project and tools.

Appendix 4: Addendum and Updates

The techniques described in this paper were adapted for use at other clients.

On a recent project, the system under test was a multiple-user system with complex clients and an object-oriented database on a shared server. The system was developed by a team using approaches from Extreme Programming and DSDM methodologies. Although the organisation was small, the testers worked for the 'commercial', rather than the 'technical' side of the company.

Session-based testing was found to be a good way of controlling and directing a team made up of one consultant, and a variety of (mainly inexperienced) testers. Although the team composition changed daily, session logs proved an effective means of recording activities and events for later use by different people, and allowed stability and repeatability despite the rapid pace of change. Sessions allowed the experienced team members to direct and limit the exploratory activities of inexperienced testers, and allowed specific testing skills to be coached rapidly into the team. Sessions were also useful for controlling and recording tests with 6-8 testers operating simultaneously on the same test from different rooms. The interaction and empowerment of session-based exploratory testing was identified as a key part of the good morale within the test team.

Pitfalls revealed during this project included:

- Where a test session is used to examine multiple test points, the calculation of coverage needs to be refined. A spreadsheet is not a scalable solution to this problem.
- A spreadsheet is not a scalable solution for holding historical information about past sessions (statistics, not logs).
- Using a spreadsheet to link bugs to the sessions in which they were found is useful, but difficult.
- Careful analysis and description of Test Points is necessary to effectively direct inexperienced testers. A review may be needed shortly after the start of the session, as well as at the end.
- While sessions were a good way of recording tests and problems, bug ownership was more problematic. Problems could often be clarified from sessions, but intermittent problems were harder to promote convincingly.
- For coaching/direction reasons, it may be difficult to scale a team beyond three inexperienced testers to each experienced tester, when all are working at the same time.
- Inexperienced testers can learn effectively from exploratory, diagnostic tests of bugs they have discovered, but such tests can exceed the time spent on the original test session. It is important for tracking purposes to log another test point/session for diagnostic tests. This also allows diagnostic tests to be prioritised with other, possibly more urgent, test points.

Further tools to track and report on progress were developed in Excel. These tools have been refined, and a project is underway to re-implement them in PHP and MySQL for web deployment.

A tiny, pretty, Flash-based tool for use as a session timer is currently available for free.

Tools and further updates may be found at:

http://www.workroom-productions.com/papers.html