# From a Sow's Ear to a Silk Purse

## Making the most of what you've got in software testing

**Abstract:**

The test team has a greater influence on the success of testing than any single process, tool or technique. Yet, under-resourced and over-stretched, it can be a source of weakness. This paper outlines some effective techniques to help get the best out of your team, without increasing your budget or finding fewer bugs. Staring with an approach that enables a team to find and set aside time to spend on process and skills improvement, the paper looks at automating common tasks and using more efficient and effective test techniques. After outlining the characteristics of a process of continuous improvement and ways to influence it with effective communication, the paper concludes with a brief study of cheap tools and web-based information sources.

**Author: James Lyndsay, Workroom Productions Ltd.**

James is an independent test strategist, based in London. He's spent more than 10 years in software testing and has been the principal consultant at Workroom Productions since its formation in 1994. As a consultant, James has encountered a wide range of working practices, from rapidly evolving Internet start-ups to traditional large-scale enterprises.

James is a director of The Manual (http://www.the-manual.org/), a not for profit organisation dedicated to gathering and publishing basic skills.

# 1. Introduction

The test team has a greater influence on the success of testing than any single process, tool or technique. Yet, under-resourced and over-stretched, it can be a source of weakness. While the team can be strengthened by automating some tasks and using effective techniques, long-term strength and resilience comes through encouraging a process of continuous improvement supported by effective communication. Test teams should look to do *better* testing, not *more* testing.

# 2. Finding the time

Software is complex. Making it is a creative process. Internal deadlines creep towards immovable external limits, and within all this, testing can be a weak solution to an open-ended problem. There never seems to be enough time to test adequately – let alone to pause long enough to make improvements.

However, time-savings can be found. Like money, it is easier to fritter away little bits of saved time than it is to put it aside for later use. By recording the time that has been saved in a time bank, it can be ring-fenced for later use.

## 2.1. Common time savings

Even efficient, well-run teams have opportunities to find time here and there – perhaps as a result of a one-off event rather than a significant, systematic saving. Often, the most immediate way to save time is to cut down on time spent in meetings – five or ten minutes here and there can soon add up.

### 2.1.1. Meetings

Although communication is vital (see below), some time spent in meetings may be better spent on another task. It is worth looking out for bad habits; a habitual length, habitual participants, the same issues discussed in the same way can all be pointers to a self-sustaining meeting. There are a variety of techniques that can be used inside and outside the meeting to reduce the time spent and concentrate on effective communication:

- Occasionally **vary meeting lengths and frequencies**. If a shorter or less frequent meeting is still useful, it may be possible to make the change more permanent. Try doing a 30-minute meeting in 20, or a 60-minute meeting in 45.
- If your meeting commonly overruns, or starts late, recognise this and **offer incentives for a prompt start or finish**. Having staggered finish times can allow a specialised sub-group to focus and carry on alone.
- Introduce techniques to **move the meeting onward** if it gets bogged down. In particular, if an issue causes conflict between some, but not all of the participants, it may be worth resolving outside.
- If the meeting has a **few active participants**, and many people who simply attend, consider a different approach.
- Occasionally **review the purpose and scope** of the meeting. If it is trying to fit too many requirements, consider splitting it into smaller, shorter meetings. It may also be possible to bring together two meetings with similar purposes and participants.

### 2.1.2. Streamlining and Drudgery Avoidance

Existing processes can be streamlined, and dull, repetitive processes can be automated. However, this generally needs some time invested in analysis and tool creation. If there is an existing effort to sort a process problem out, then the time saved is available for banking. If not, it may be worth waiting until some time has been banked before attempting to gain more time by improving a failing process.

### 2.1.3. Pitfalls

Take a moment to verify that the time you're saving is a genuinely available. It's easy to try to grab extra time, rather than find time that could be better spent elsewhere. If you take time from otherwise effective tasks, you may reduce the effectiveness of those tasks – and spend more time sorting out problems than you gained in the first place.

Note also that unless the team is motivated to spend their own free time on finding improvements, time spent during lunchtimes and working late is time stolen, not saved.

## 2.2. Using a time bank

Make your team aware that you're trying to find some time – and that the time found will be set aside for making things better, not for more testing or management. Give them a date on which you'll use this time – preferably a date when a more urgent task won't get in the way, so avoid deadlines and delivery dates.

Somewhere the whole team can see it, write up an accumulating total of time saved, over a period leading up to your set date. A week works well. Five minutes saved by one person goes up as five minutes; five minutes saved by ten people goes up as fifty. On the appointed date, use the time – preferably take the whole team away from the test area and its distractions for as much time as you've banked, split between the team. If your team of six have banked an hour over a week, that's ten minutes – but the duration is less important than the opportunity to redirect the team's attention from the tasks at hand to longer-term goals.

Initial use of time from a time bank might be to identify a process improvement or a tool that would help save / find more time; the simpler, the better. Time accumulated in the bank after this identification can be used to achieve this saving, and the saving itself banked for further use.

A time bank is, of course, an illusion – time saved can't be banked to be withdrawn later. However, the metaphor can help to isolate time away from day-to-day activity, and can help justify the use of the time both inside and outside the team. The gradual increase of time in the bank can give the team a sense of ownership, and a reward when they take the time to improve their processes and skills. Finally, because it is time saved, rather than time taken from productive tasks, it is perhaps easier to take a greater risk with the return on the investment of time.

You may find that a time bank is a useful tool to keep running – and it can certainly provide interesting metrics. However, habit can make it less effective and it is less intrusive to introduce a time bank as an agent, and enabler, of change.

# 3. Automation

There is more to automation than capture/replay tools. Introducing simple automation not only saves time, but can improve accuracy and flexibility. Good places to look for tasks that can be helped by automation include:

- Tasks that are prone to 'finger trouble' errors – i.e. setting up an interrelated data set for a test.
- Tasks where the same data, or variants, are used many times – i.e. tests against different configurations.
- Tasks where finding out something about a test involves poking around inside a database – i.e. looking for all data that *should* have been retrieved.
- Tasks where many similar things have to be compared, looking for one rare difference – i.e. checking environment changes.

Some tools are best built from scratch by the test team (spreadsheets are a good platform for this). However, it may be more productive to spend time looking at open source tools, or tools re-purposed from elsewhere in the organisation.

Some useful ideas for testing tools include:

- **Data load and examination tools.** Most databases come with tools that allow tables to be loaded from external data sets. Being able to reloaded tables with clean data can help reduce test errors, while tools that allow the data structure and contents to be queried are vital to diagnostic work and aiming re-testing. *TOAD* (Oracle), *PHPMyAdmin* (MySQL) and *PHPPgAdmin* (PostgreSQL) are examples of tools that can greatly help while testing systems that use databases, while *Excel* and *Perl* are both strong tools for working with flat files. For systems that use proprietary data storage, you may be able to share tools with the coders – and if the coders have no tools, they may be grateful of a collaborative offer. It is generally possible to use the 'replay' aspect of capture/replay tools for data load if data can be input through the user interface.
- **Environment analysis.** Can compare file hierarchies, environmental variables, versioning etc. Can be as simple as a set of sort and comparison commands. Useful if run to compare version to version – but can be even more useful if run very frequently to spot unexpected changes.
- **System insight.** Static tools look at the code without executing it; some spot complexity and dead code – others allow you to strip out hard-coded values and error messages. Dynamic tools look at the system as it is working. *InCTRL5* will tell you about changes made to the file system and registry on machines running Windows variants, *Dependency Walker* will tell you about modules loaded, while *Holodeck* can tell you about processes and system calls. Some testers watch the CPU monitors and listen to the disk chatter to get an idea of what the system is doing. Network analysis tools can give an invaluable insight into how a system interacts with other entities, and log readers (of the software or the wider system) can show up errors and race conditions that can help diagnose an intermittent problem. It is often possible to use the 'capture' aspect of capture/replay tools to get an insight into the workings of a GUI or web interface.
- **Information interchange**. Issue trackers can be used to track far more than bugs (e.g. tests, customer interactions, enhancements, test process issues), and their collating, ranking and sorting functionality can take the drudgework out of many project management tasks. 'Blog's, 'Wiki's and discussion threads allow an otherwise dispersed group to discuss, disperse, record and access information in a reliable, scalable and timely way.

## 3.1. Automated metrics

Automated metrics have become simpler to collect. A web interface can be used to manually enter information into a central database as it becomes available. Some tools produce XML, or can be automated to send information. Utilities such as 'curl' (a Unix tool) can capture output from tools that have web pages, but no export facility. Information from tools as varied as test management, test execution, bug tracking and customer support can be gathered centrally.

Summaries of this project and product information can be generated in real time, and displayed to anyone with authorisation and an appropriate connection, encouraging openness and agility. This display is sometimes known as a 'test dashboard' ( a parallel concept to a 'project dashboard'). It can make prioritisation simpler, and can help the team to make decisions more rapidly and more frequently.

Of course, the value of the data is only as good as the quality of the input, and the collection and summarisation is open to flaws and misinterpretation. However, it is an important step forward from the once-a-week graph posted on a noticeboard in the kitchen.

# 4. Techniques

There are many possible test techniques, and although the choice may be limited by available skills and technology, some approaches are more effective than others. This section details a way to make some common approaches more efficient, and a way to get more out of any given approach.

Many test techniques have an associated measurement of coverage. Coverage allows the tester to measure how much of the system has been tested – high coverage allows confidence that the testers should have had a good opportunity to observe any errors that may be lurking. The most basic measure of coverage is [(completed tests) as a proportion of (planned tests)].

Many techniques ultimately arrive at test cases by combining possible options. With more than a few possible options, this can easily result in a dauntingly large planned test set, and adequate coverage can appear entirely out of reach. It is possible, however, to use an approach called 'pairwise combinations' to cherry-pick an effective test set from this huge range. This best set includes all possible pairs of options rather than all possible combinations of options, and can reduce the size of the test set by many orders of magnitude. Indeed, the larger the set of possible tests, the more dramatic the reduction. You'll need a tool to do pairwise combinations on worthwhile sets – see the appendix for link/references.

There are many ways of measuring coverage. Sadly, many are incompatible; good coverage in terms of [lines of code executed / number of lines of code] does not necessarily imply good coverage in terms of [requirements checked / requirements stated]. Furthermore, measuring done-ness as a proportion of an ideal of completion is not possible with open-ended sets such as negative tests and generated model-based tests. Retaining a coverage-based approach at all points may give a false sense of confidence, but not actually help find more problems.

Many problems are not discovered *by* a planned approach, but discovered *while doing* a planned approach. This subtle distinction can be put to your advantage, by increasing the chance of discovery. Using a variety of test machines, testers, paths through the system and data can reveal unexpected problems (ones that often cause the comment 'but it *can't* affect the system like that'). Problems can also be revealed by use of appropriate tools to give insight into the system (see 'System insight' tools in the section on automation above).

Once the failures have been noticed, testers should go through a process of diagnosis to isolate the fault. Once the fault is verified, testers can seek to discover similar faults by reverse-engineering a new test set to pick up similar problems. It may be possible to identify the error or root-cause, and address these by static tests, reviews or process adaptation.

Some techniques are used solely because they are familiar to the team, or have been used throughout the life of a product. There may well be more effective techniques available – part of the process of ongoing improvement is to become aware of possible alternatives, and to be able to assess their value.

# 5. Ongoing improvement

While tools and techniques may bring immediate, tangible gain, they don't always lead to a sustained change. This section looks at ways of introducing a strategy for ongoing improvement in the test team, leading to sustained productivity gains through increasing the range and depth of skills in the test team.

## 5.1. Training and Coaching

Software testing is a broad discipline, needing highly technical approaches supported by a good understanding of softer skills. While many general course providers cover appropriate aspects of management, negotiation and teamwork, a number of testing-specific course providers cover the more technical end of this spectrum – particularly tool use and certification (UK market).

Training is an effective, systematic way of giving people the skills they need to do a defined task. Typically, a training course is the first time that an approach has been formally introduced to most delegates, and one or two trainers take their class through a generic set of information and exercises. Training is intensive, and the trainee is lost to his or her team for the duration of the course.

However, training does not teach a team member when to use these skills, or how to be a better tester. Typically, this is learnt on the job, from other members of the team, through a process of coaching.

Coaching is an altogether more intimate relationship, with an emphasis on tailored feedback and highly specific challenges. The tester is often encouraged to follow their own path, rather than follow a pre-defined route, and coaching tends to be an ongoing, long-tem and intermittent process of small-scale intervention.

Coaching is a very effective way of getting more from a team, in particular by growing those skills that are directly focussed on the project in hand. Coached skills tend to be learnt by practice and example, and so are harder to forget than skills learnt in a classroom or lecture. The activity of coaching can be rewarding to both participants, and encourages respect and communication within the team – and the roles are often reversed for different skill sets. It reduces the dependence on a single person's skill set by distributing their skills more widely. Indeed, as coaching is an effective way to leverage experience within a team, it may be appropriate to bring a person into a team on a temporary basis for the express purpose of working with the team to bring their skills and processes to a different level. This is particularly useful when moving to a new strategy – from manual testing to a more automated approach, from scripted to exploratory, or vice-versa.

Coaching needs to be actively encouraged. Without active encouragement, team members may not feel they can take the time from their stated, urgent tasks to coach each other. Indeed, coaching is itself an acquired skill, and some experienced people in the team may not feel comfortable coaching their peers – or being coached. From the point of view of time spent, it is important to realise not only the benefits that accrue, but also that most coaching happens while both participants are doing their jobs, and that they continue to be productive.

## 5.2. Learning alone

A motivated team member will want to extend their skills beyond those immediately available through training and coaching. This should be encouraged, and a wide variety of sources of information are available.

A shelf-full of testing books costs less than the average ISEB certification course or a day of consultancy fees – and while they may not be in constant use, they can provide a deep well of knowledge and experience when needed. To encourage learning as well as use, it may be necessary to provide a quiet space, or self-study time.

Software testing has few periodicals. *STQE* is perhaps the most widely read in the UK and US, and *Professional Tester* is a lower-cost, well-distributed UK magazine. Wiley's more *Software Testing Verification & Reliability* is more expensive, and more academic.

The Internet allows access to an unprecedented range and depth of practical information about software testing. As with all information on the net, much of it has to be taken with a pinch of salt, but it is possible to not only discover unconsidered alternatives, but also to interact with consultants, experts and practitioners. Software testers can feel isolated in an organisation; the Internet provides an opportunity to become part of a genuine community. Details of Internet resources are given in the appendix.

## 5.3. Team learning

As individual team members gain knowledge about test techniques, it may be productive to encourage them to present this information to the team. This not only spreads information, but encourages discussion and an active interest in application of the techniques. Team learning can also be highly effective when then system under test is changing rapidly, as individuals who have worked on changed areas can present a focussed description of the changes to their team. It is best to keep these presentations brief and informal, and to allow in-depth discussion between interested parties to happen off-line.

Teams can improve shared skills such as estimation and bug logging if the processes that record the results of those skills have a fast feedback between action and effect. For instance, revealing the difference between estimated time needed and actual time taken at the point where the time taken is recorded allows both the designer and executer of the task to consider any differences – and perhaps work toward improving the accuracy of the next estimate.

## 5.4. Supporting the process

Feedback is vital when developing a process of ongoing improvement. Some feedback is neutral, and may be automated – a test dashboard showing overall progress in real time is an important tool not only

for planning and reacting, but also to help the team learn. Some feedback is more personal; reviews of activities and deliverables are necessary to allow improvements to be identified and prioritised. Reviews can be uncomfortable, and the reviewer should take care to review the product rather than the person.

Motivation is important to learning. A happy team generally learns with less resistance, and retains the learnt information for longer.

Individuals who feel responsible for a particular task are more likely to want to improve that task, and giving responsibility through genuine ownership is a powerful tool to arriving at an improved piece of work. Genuine ownership is supported by a management approach that is hands-off and nurturing by turns, and it is vital to let the owner feel that they are empowered with the time and support to change the process if needed.

Not all testers have an appreciation of the wider world of testing. Encouraging team members to interact with testers outside their team can give renewed enthusiasm and an understanding of the scope and variety of possible approaches. Wider interaction can also encourage team members to see a viable career in testing, which may itself increase motivation and promote self-improvement.

A team that wants to steadily improve its work has to recognise that the improvement is itself a piece of work, and needs time and energy devoted to it. Mistakes are a necessary part of the process, as is moving on from established practices – and some of the time spent will be spent badly. Using a time bank, or a budgeted proportion of time spent on other activities, can help to keep the process going through troubled times.

# 6. Communication

Effective communication allows ideas and problems to be shared and allows the team to work to the strengths of its members. Some teams already communicate well. Others regularly work under headphones, indulge in the well-known “us and them” attitude, and don’t share work. These teams are more likely to log duplicate bugs, and less likely to pull together toward improving their work.

Some ideas to encourage effective communication include:

- **Testing in pairs** is a great way to see new bugs and techniques, and encourages an effective working relationship between team members.
- A **testing dashboard** showing real-time metrics and progress not only provides important feedback to a team, but can encourage other teams to trust the testers. By communicating information in real-time, it is seen as ‘un-fakeable’, and gives an insight into the cycles of work that characterise testing.
- Communication **outside the team** can give a greater understanding of the business and its customers. It can also forge political and personal links.
- **Coaching** and **peer-presentations** can help individuals to feel more at ease in making their voices heard – and can encourage the team to both respect and question the views of their peers.
- **Bug reviews** not only reveal hidden patterns of problems, but can bring the team together in an understanding of characteristic flaws. It may also help to improve logging standards.
- **Suggestions**. Suggestions can be large and small, may be anonymous, and should be reviewed and discussed rapidly and regularly. Most suggestions will be discounted, but should be kept to generate ideas later.
- Instigating a **regular, budgeted meeting** for the whole team to discuss the current approaches and ways that they could change or be improved. Shorter, more frequent meetings can help the team avoid getting blocked by a single issue.
- A **ten-minute daily meeting** to discuss today’s plans and yesterday’s events.
- **Lunches together**. You may want to ban ‘shop’ talk at these lunches – or they could be sponsored by the company if they are to take a discussion out of a formal meeting (see note above about stolen time).

# 7. Low-cost, high-value resources

Software testing is a specialised discipline. Some tests cannot be done reliably without expensive tools – but it is worth recognising that the fragmented nature of the industry does not encourage efficient competition. Some tools are over-priced, and under-used. Some are valuable, and inexpensive.

It should be noted that the greatest expense of tools lies in implementation and training, and not generally in licensing cost. However, cheap tools allow the test team to experiment with approaches and discover potential without having to justify capital expenditure.

## 7.1. Tools

Microsoft *Excel* is the Swiss Army knife of software testing. Other spreadsheets and similar tools exist, but *Excel*'s flexibility and ubiquity makes it the key customisable tool for many testers. Most companies have at least one copy installed and paid for.

A modicum of expertise in *Excel* can pay huge dividends in:

- Data generation, data manipulation
- Test case generation
- Test management and tracking
- Defect management and tracking
- Data analysis
- Metric collection, summarising and reporting

Testers can be greatly helped by knowledge of the following areas:

- Effective data labelling
- Data filtering
- Mathematical and statistical formulae
- 'Array formulas'
- Pivot tables
- Absolute and relative cell referencing
- Scenarios
- Conditional formatting
- Conversion of columnar data to comma-separated
- Parsing text data to columns
- Graphing

It is important to remember that *Excel* is no substitute for a database. For test and issue management, a tool that handles one-to-many relationships is a better fit than a list handler, and a relational database offers a different kind of power. Microsoft *Access*, *PostgreSQL* and (open-source) *MySQL* are all capable and cost-effective/free databases. *MySQL*, in particular, is fast, scalable (within the needs of most test teams) and portable.

It is becoming simpler to write dynamic web pages for internet/intranet use. Again, open-source tools such as *PHP*, *Ruby* and *Python* allow experimentation without investing in a tool up-front. They also attract coders who develop generalised libraries of common functionality.

Defect trackers used to be expensive. Now, tools such as *Bugzilla* offer a powerful open-source alternative for self-hosting. Internet services such as *Elementool* offer instant access to a robust, generalised, remotely hosted system – and often offer to manage a number of issues for free. Payment, when necessary, is monthly and can be based on a per-project cost, rather than a per-seat. If you plan to build on these services, ensure that your valuable data can be exported for re-purposing.

Macro players exist in a variety of forms on many platforms, and are usually cheap. If they have the capability to perform conditional logic on data drawn from a file, it is possible to use them for automated testing and data load. Note that it may be far simpler to re-use data-driven tests with a different tool that it is to re-code scripted tests. Most capture/replay tools are expensive. However, *Vermont High Test* is a tool priced under $300 that offers many of the capabilities of more expensive tools.

Many tools can be re-purposed and used for testing. *InCtrl5* was designed to help users see the changes after installing a new package to a Windows PC. It is a great tool for testing installation, upgrades, uninstalling – but is also useful to spot unexpected changes that have been caused by normal use of a

system. Danny Faught has put together a list of free / open-source tools that can be used for testing, and James Bach's paper 'Improving your testing superpowers' is another useful reference. Note that licensing issues and your company's software policy may mean that use of open-source tools may need be restricted or partitioned to non-production systems.

*Holodeck* is a (windows-based) tool that lets you monitor and affect the interactions of an application with the platform it is running on. It has been shipped as a free tool on two books, is available for download, and allows an unparalleled insight into the detailed workings of a running application.

## 7.2. Information

A large amount of information is available on the internet. Much of it is available for free or at a nominal cost.

Participating in a discussion group, a moderated forum or one of StickyMinds' roundtables can expose an otherwise isolated tester to a wide variety of business practices and test possibilities. Discussion groups are generally free to join, and are frequented by consultants and subject-matter experts. They are excellent places to get focussed suggestions to intractable problems. For instance, the tool-specific groups on Betasoft's QA forums allow your team to not only discuss common problem with other tool users, but also gain an understanding of the real value and uses of the tools.

Many consultants promote their practice and extend the discipline of software testing by publishing papers at conferences. These papers are usually available in the conference proceedings – but many consultants also make these papers available from their own websites. Some consultants write newsletters, 'Blog's or brief articles published only on the web. Tool vendors and larger service providers also publish white papers, which are generally available for free or as a swap for your email address.

Some websites act as information hubs. These generally provide links to further websites, but some have content of their own. StickyMinds is the largest of these – it allows some general use, but offers enough incentives to make its 'PowerPass' full access an attractive option. An interesting new site is testingeducaction.org, a website set up with a grant from the National Science Foundation of the U.S.. It contains papers and course notes from a variety of consultants, with a particular emphasis on those aligned with the 'context-driven' school of testing.

# 8. Conclusion

Changing circumstances, skills and constraints introduce a variety of opportunities for improvement. Without a policy of ongoing improvement, these opportunities can be lost, and the existing skills and efficiencies of the team threatened. An effective team needs to actively seek and encourage improvement.

To take advantage of these opportunities, teams – and the individuals that form them – need to be able to experiment with different approaches. It is easier to justify these excursions if time has already been set aside to look at viable alternatives to existing practice. This time needs to be budgeted for, or found.

Finally, it is often easy to see an ever-increasing need for testing. Rather than being overwhelmed by the volume, it is better to interpret this as a need for better testing, not more testing.

# 9. Appendix and references

*Note – the author has no commercial links with any of the tools, vendors, consultants or service providers listed in this paper. All recommendations are based on personal experience, and you should assess your own situation before taking any decisions. Your mileage may vary.*

See http://www.workroom-productions.com/MakingMost.html for updates to this list.

## 9.1. Web links for tools

- Testing FAQ's Tools list (see descriptions for freeware/open source): http://www.testingfaqs.org/
- Opensourcetesting.org's list of, you guessed it, open source testing tools: http://opensourcetesting.org/
- Danny Faught's initial article (small list; list of 100+ not yet publicly available): http://tejasconsulting.com/DFWUUG/freewaretools.html
- Danny Faught's recent article on StickyMinds: http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=6454
- James Bach's cheap tools article: http://www.satisfice.com/articles/boost.htm

**Direct links to tools.**

- Elementool (issue tracking service, first 200 issues free) - http://www.elementool.com/
- InCtrl5 - http://www.pcmag.com/article2/0,4149,25126,00.asp - note: no longer free (!)
- Vermont High Test (cheap-ish capture/replay) - http://www.vtsoft.com/
- Bach's 'pairwise' tool can be accessed by a link from http://www.satisfice.com/testmethod.shtml
- Holodeck http://www.se.fit.edu/projects/holodeck/

## 9.2. Web links for information

**Test information hubs**

- StickyMinds http://www.stickyminds.com/
- Testing Education site http://www.testingeducation.org/
- Software QA Test Resource Center http://www.softwareqatest.com/index.html

**Magazines**

- *STQE* http://www.stqemagazine.com/
- *Professional Tester http://www.professionaltester.com/*
- *Software Testing Verification & Reliability* http://www.interscience.wiley.com/jpages/0960-0833/

**Consultants with available papers / active websites (small selection)**

- James Bach (Satisfice): http://www.satisfice.com/
- Cem Kaner (F.I.T.): http://www.kaner.com
- Elizabeth Hendrickson (Quality Tree): http://www.qualitytree.com/
- Brian Marick: http://www.testing.com/
- Bret Pettichord (Pettichord Consulting): http://www.pettichord.com/
- Robert Sabourin (AmiBug): http://www.amibug.com/
- Esther Derby (Esther Derby Associates): http://www.estherderby.com/

**Discussion Groups (small selection)**

- QA Forums from Betasoft http://www.qaforums.com/
- comp.software.testing (on google groups)
- StickyMinds roundtable http://www.stickyminds.com/roundtable.asp?tt=RoundTables

**Pairwise testing paper**: The Combinatorial Design Approach to Automatic Test Generation (IEEE Software September 1996, pp. 83-87) – Cohen, Dalal, Praelius, Patton
http://www.argreenhouse.com/papers/gcp/AETGissre96.shtml