2009

PACIFIC NW

SOFTWARE QUALITY CONFERENCE

MOVING QUALITY FORWARD

OCTOBER 27-28, 2009

Conference Paper Excerpt

From the

CONFERENCE PROCEEDINGS

Too much automation or not enough?
When to automate testing.

Keith Stobie
Keith.Stobie@microsoft.com

Abstract

Fundamentally test automation is about Return On Investment (ROI). Do we get better quality for less money by automating or not automating? The obvious and famous consultant answer is "it depends". This paper explores those factors that influence when to choose automation and when to shun it.

Three major factors you must consider:

1) Rate of change of what you are testing. The less stable, the more automation maintenance costs.
2) Frequency of test execution. How important is each test result and how expensive to get it?
3) Usefulness of automation. Do automated tests have continuing value to either find bugs or to prove important aspects about your software, like scenarios?

Biography

Keith Stobie is a Test Architect for Protocol Engineering team at Microsoft working on model based testing (MBT) including test framework, harnessing, and model patterns. He also plans, designs, and reviews software architecture and tests.
Previously Keith worked in Microsoft's Windows Live Search live.com and XML Web Services group. With twenty five years of distributed systems testing experience Keith's interests are in testing methodology, tools technology, and quality process. Keith has a BS in computer science from Cornell University.

ASQ Certified Software Quality Engineer, ASTQB Foundation Level
Member: ACM, IEEE, ASQ

1 Introduction

Books on Test Automation (Dustin99, Mosely02) usually spend a few pages on the question of whether to automate. Fundamentally test automation is about Return On Investment (ROI). Do we get better quality for less money by automating or not automating? The obvious and famous consultant answer is “it depends”.

Three primary drivers are cost to create, run, and maintain, but there are many other differences between automated and manual tests. Further, it is not an all or nothing choice. Automation can be done for some test cases and not others. It can even be done for parts of a test case and not others.

More subtle is the realization that an automated test is rarely the same as a manual test. A manual test is performed by a human with human capabilities of observation, and ideally given the charter to improvise to enhance its value. An automated test is usually limited to what its creators thought about and coded. Other characteristics of manual vs. automated tests, such as metrics, are not considered in this paper. Suffice it to say that you get different data from these two types of tests with varying levels of accuracy. If the metrics and accuracy are of primary concern, and not cost, then these other factors must be considered.

2 Why Repeat

Repetition is the first key question about when to automate testing, but not the only question.
Sheer repetition is typically sufficient to get ROI for automation. However, you must understand why you need the repetition to justify the investment. Repetition for no reason is just a waste of money.

If there were no defects to be found, there would be no testing needed. Testing provides information to give confidence, but if you knew there were no defects, that information would be of little value. I don’t know of any large, modern, complex software systems where anybody knows there are no defects. When doing conformance testing, the defect being checked for is lack of conformance.

The frequently stated reason for repetition is: any repetition has the possibility of a finding a defect if something has changed. That something could be the inputs, the software, or the environment. While it is true that almost anything has the possibility of finding a defect, testing is the art of choosing those activities having the greatest possibility of finding defects of the most consequence. Choosing which defects are of most consequence depends on context. A typographical error in an error message for a computer game may be of little consequence, while that same error in a log message for enterprise operations to act on could cause operators to take actions resulting in the loss of the system.

Most customers are more upset when a previously working feature fails than when a brand new feature fails. But if the brand new feature is the reason for purchase and not the old features, then new features working becomes the more critical factor. Thus the consultant’s statement “it depends”. You must know what is more important for your customers.

Whether repetition helps find defects of the most consequence will significantly impact your choice to automate testing. Choosing what to repeat is critical. The cost of repetition also factors highly into the equation.

2.1 Change the input values – data driven

As an example, in designing a test case to verify the square root function for a 32 bit computer the following options are available:

If you are worried about testing the function

- just once with one input,
  then manual testing will be most cost effective.

- with a few sample points only a few times,
  then manual testing may be most cost effective.
- with a few sample points, but many times,
  then automation becomes imperative.
- completely, even just once, then automation is imperative. Verifying 2^32 inputs and their corresponding output by a human will vastly outweigh the cost of automation.

Testing a simple square root function has the cost of automating small relative to the cost of manual verification. Even testing a 64 bit computer's square root function is practical assuming you have over a million computers (say 2^20 spread over the internet by volunteers), and each value can be tested in a nanosecond, resulting in about 2^30 values tested each second on each computer. Then only 2^14 seconds or about 4.5 hours are needed to verify all possible inputs and resultant outputs.

However, just changing from the simple world of integers (or floating point numbers) to strings, makes the all values testing impossible. Each Unicode character in a string is roughly one of 2^16 characters. If I have 8 characters, then there are 2^128 possibilities. With the same resources above, it would take 4.5 hours * 2^64 or about 10 quintillion years. So, except in rare cases, verifying all values is not possible.

2.1.1 Value Selection

How many values to sample to give confidence and which values are most likely to find defects of consequence is the subject of many testing books and papers (e.g. Whittaker02). This paper presumes you know how many values you need to sample. A difference between automated and manual tests is the strategy for choosing values of consequence. In either case you can fix the set of values ahead of time (perhaps chosen based on boundary value analysis or other techniques), which makes it merely a question of repetition. With automated or manual tests you can provide different dynamic strategies. Automated tests may use a random or adaptive random testing strategy. Exploratory testers may use intuition and past observations.

2.2 Changing the environment – setup

Besides changing the input values, another area to vary is that of the environment. You may want to run using different versions hardware or of related software including different operating systems, languages, etc. The setup costs for these different environments will drive the choice for automation. Many companies are using Virtual Machines as a means of lowering setup cost. If tests must be repeated on different platforms, versions, languages, etc., then all of those repetitions increase the frequency of test execution. Making automated tests configurable to handle the different platforms, versions, languages, etc. may also have an added cost.

For example, one company may need to just sample a few times whether their software works when a USB device is unplugged and plugged back in. Manual verification is probably their cheapest solution. Another company needs to verify many pieces of software against many different USB devices being unplugged and plugged back in. The second company may find a $100,000 investment in machines to mechanically plug and unplug USB ports cheaper than hiring manual testers to do the same work over many months or years.

2.3 Changing the software – regression

The third area of change, the software, depends on the nature of the change. Is the change a refactoring that is not supposed to affect the external behavior of the software? Then automated testing becomes more feasible because the expected outputs are not expected to change. Is the change a new aspect of the software that cuts across many features and changes their external behavior? Then either manual or automated testing will have to account for the new behavior. The manner in which the manual or automated testing is performed will affect the cost of accommodating the new behavior. Many books about software, testing and test automation deal with various technologies to reduce the cost of accommodating change (e.g. Gamma94). This paper presumes you have an understanding of your costs for accommodating various types of changes.

See also the Test Maintenance section later for further discussion.

3 Latent vs. Regression defects

After a test with specific input on a specific environment, against a specific software build or version shows the software behaves as expected for that input and environment, then there is virtually no possibility that the test will later show the software does not behave as expected, unless the input, environment, or software changes.

If a defect exists in your software, but has not been discovered by your testing process, it is a latent defect. Changing the inputs or environments without changing the software may reveal where the software does not behave as expected for the current build or version. This can show latent defects.

Changing the software to a new build or version, without changing the inputs or specific environment may reveal changed outputs. If the changed outputs are expected, then tests require corresponding changes. If the changed outputs are not expected, this is called a regression. A regression is an unexpected change in output due to a new version of software. The unexpected change in output is that it doesn't match the output of the previous version.

From a zero based budgeting perspective, you have at every point in time the following choice: look for more regression defects or more latent defects. At any time your software is at a state where you've identified (and hopefully usually removed) known defects found by your testing process. Specifically you can repeat an existing test on changed software, or you can run a new test on the software (changed or unchanged). If your software hasn't changed and you have run all the regression tests you desire and you have remaining time and budget, then latent defect testing is the obvious way to go. However, if you don't have time and budget to run all of the regression tests you desire (almost always), then you have the hard choice of whether to run a regression test or a new test.

Many test managers focus almost exclusively on regression testing just before shipping. If regression testing finds the most defects of greatest consequence, then this is a wise choice. As stated earlier, a business context determines whether finding regression defects is more important than finding latent defects.

The context in which your software operates is critical. Safety critical software that affects human life costs more because the testing must be more thorough, repeatable, and justified. Similarly software that has significant financial impact is usually funded for more testing. On the other hand, a small custom application for a single department of 20 users doesn't have the same funding for testing, nor the same need for automation.

Running a test the first time, whether automated or manual, is about finding latent defects (existing defects missed earlier). Running the same test again with only new software is usually about regression defects (new defects unexpectedly introduced). Over time concern frequently shifts from latent (incomplete product testing) to regression (changed product) defects, and thus manual testing frequently shifts to automated testing.

Numerous studies and papers have shown that detecting and removing defects early saves money. Thus detecting latent defects and regression defects early saves money. Typically manual tests are most cost effective for finding latent defects early. If the manual test finds the behavior expected, it is not repeated. Typically automated tests are most cost effective for finding regression defects early as they can more economically be repeated after frequent and small changes.

Note that even after you have automated tests, regression testing is not free. If any unexpected outputs occur they must be analyzed to determine if the test software or the software under test (SUT) is at fault.

Both being software, they are both likely to have defects resulting in the unexpected result being observed. See Examples of Automated and Manual analysis for further discussion.

3.1 Capture/Replay

A frequent, but sometimes dangerous way to change from manual tests looking for latent defects into automated tests looking for regressions is to record the manual test in some fashion and then be able to use that recording as a way to repeat or replay the test. This can be done at any level, code, User Interface (UI), protocol on the wire communication, etc. This major issue is the assumption that a manual test should be transformed into an automated test. Many latent defect finding tests aren't worth repeating. Their value in finding a latent defect was perhaps marginal and repeating them provides even less incremental value.

Given that automating a test is more expensive than running a test manually, which regression tests to automate should not default to all of the attempts made at finding latent defects.

4 Manual vs. Automated tests

Human versus computer execution is emphasized in lesson #108 in Kaner01 "Don't equate manual testing to automated testing". Manual tests can also be frequently described less exactly than automated tests. To have an automated test, it must ultimately result in code executable by a computer, and most instructions to computers today are fairly exacting with formal syntax and rules, whether a scripting language such as Ruby or Powershell, or a compiled language such as Java or C#. Automated tests using keyword or action word automation elevates the abstraction language, but is still ultimately turned into a detailed sequence by software.

4.1 Exploratory Tests

Manual tests may have only broad charters that employ exploratory techniques of concurrent iterative plan, design, execution, and evaluation of test ideas. Exploratory testing is effectively only possible today using humans doing some form of manual testing. Some types of testing are best done using exploratory testing. Usability testing in terms of how humans perceive an interface generally requires humans. You can perhaps use automated tests to verify user interface guidelines, but not necessarily usability.

4.2 Scripted Tests

The other form of manual testing can be called scripted testing that describes step by step procedures. Still the level of detail of the steps can vary widely. It might be high level and loose, for example:

Login
or Enter username and password

Or low level and very tight such as

Click mouse in username field to place cursor in username field. Type "testuser".
Click mouse in password field to place cursor in password field. Type "Pa$$w0rd"
Click on login button

The second script is approaching the level of detail eventually required for an automated test (although an automated test might also have high level methods like

```
LoginEnteringUsernameAndPassword(string username, string password)
```

This results in execution similar to the detailed script.
The first script allows the human, manual tester, a great degree of freedom in how to approach the task. They can choose the method of entry: keyboard short cuts? Tabbing to fields? Using the mouse? They can choose the data to enter: if multiple usernames with passwords are available they can use different ones at different times.

When scripted tests use high level loose steps, they are cheaper to create and cheaper to maintain than low level tight steps. However the initial execution cost of high level loose steps can be higher since it requires more domain knowledge. Assuming the same human (or set of humans) repeats the tests within a few days or perhaps weeks' time frame, then repeating a high level or low level script is roughly the same cost. The cost for high level running goes up when domain knowledge is lost due to change in personnel or infrequency of repetition.

4.3 Manually written tests

The cost of automating tests can vary widely also depending on tools and technology available and implementer knowledge. Automated tests are just a form of software and studies have already shown a 10 to 1 factor in skill level of programmers. That is, a very skilled programmer can, in one hour, create as much functionality as a poorly skilled programmer does in ten hours.

Most automated tests today are created by hand coding. They may be specified by a subject matter expert using keyword driven testing and automated by a specialty test interface team. They may be written by software engineers with only a cursory knowledge of the domain. In most cases defining and automating a test (or set of tests) and executing it once is more expensive than defining and manually executing a test (or set of tests) once for the same functionality.

4.4 Automatically generated automated tests

Instead of creating each test by hand, tests can be automatically generated. Simple methods of automatic generation include data driven tests. More advanced test generators are frequently based on formal models such as grammar models or behavioral models (such as state diagrams).

4.5 Test Repeatability

Using either detailed scripted or automated tests we can get highly repeatable tests. Automated tests are typically more repeatable as most humans have a difficult time exactly repeating detailed instructions (for an exception see Saran08).

Test automation is ideal for showing that for a few data points the software behaves the same as before. This can come either from manually written automated unit tests or from integration tests to show a software build appears as stable as a previous build. These are frequently called build verification tests.

4.5.1 Unit Test Repeatability Essential

In the Agile community I heard stated that all code written without automated unit tests is legacy code. Legacy code is the old, decrepit code no one wants to touch for fear of breaking. If you don't have unit tests, the risk of changing code and inadvertently introducing a defect increases significantly. When deliberately introducing a breaking change, the chance of side-effects is even higher. Unit tests make code changes safer. Automated unit tests usually make code safer than manual unit tests as they are more repeatable (does it appear to "work" as well as it used to) and are normally run more often.

Manual testing is rarely useful for repeatable unit testing since it is very low cost to transform most tests at the unit level from manual tests to automated tests, and running them frequently provides the early regression capability needed. The cost of manual execution of unit tests usually causes them to be executed less frequently and degrades their ability to act as early regression tests as further described in Marick00.

4.6 Test Maintenance

Don't underestimate the maintenance problem. Because of maintenance, often tests are picked because they are easy to maintain for automation, not because they will necessarily result in the greatest increase in product quality. For example, customers might use a mouse most when interacting with a GUI and the customers might find numerous issues because it was easier for the test team to automate the GUI with keyboard short cuts rarely used by the customer base.

Maintenance costs for automated tests, like software, depend on the original design and architecture. Software design patterns and many other software techniques have adaptability as one of their concerns. If the software was not written to be adaptable, but must change, the cost of test maintenance can be quite high. If the software is written to be adaptable, the cost of change can be quite low.

I have seen the same for automated test suites. Badly designed and architected GUI automated tests are very brittle. I've seen a test team say it would take 2 weeks to update their tests for a change to the product software that took less than an hour to design, create unit tests, code the change, and verify it with the unit tests. I've also seen a test team indicate it would take 15 minutes for them to update their test software due to a change that took 2 days to implement in the product software.

If you automate UI testing early and the UI changes every day in multiple ways, the best case is you keep up with the UI in your automation. The worst case is you quickly fall hopelessly behind.

5 Return On Investment

Testing is a business activity with a cost. To calculate the ROI for test automation, you must be able to measure the quality of the software being tested. This is a topic unto itself addressed in other books and papers. Secondly, you must measure the cost to get that quality. As Hoffman points out (Hoffman99) there are many intangible costs that are difficult to realistically assess. The goal with test automation is increased quality for the same cost or reduced cost for the same quality.

Using your resources wisely to most effectively cover the risks requires:

- understanding the product and release goals
- dealing with common issues like known errors
- anticipating likely errors
- choosing the right evaluation methods

In doing any ROI calculation, the high degree in human variance must be allowed for. That is, it may not be determinable in many cases whether manual testing or automated will work better since other factors such as personnel skill and technology known could dominate the cost.

Assuming your organization has determined the level of script detail it wants and alternatively what software technology it will use with which skill level of staff, then you can begin to do some comparisons. Frequency of test execution *alone* is not a good measure, as given by lesson #109 in Kaner01, "Don't estimate the value of a test in terms of how often you run it."

Related to the ROI is the opportunity cost of manual testing versus automation. Initially, for most types of testing, you can test more once manually than via automation. But the more you retest without having to do significant maintenance or alterations, the more attractive automation becomes. Marick expresses this (Marick98) as:

> If I automate this test, what manual tests will I lose? How many bugs might I lose with them? What will be their severity?

Three major factors you must consider:

1) Rate of change of what you are testing. The less stable the SUT, the more automation maintenance costs over loose scripting or exploratory testing.
2) Frequency of test execution. How important is each test result and how expensive to get it? Running all tests all the time is not economically feasible.
   It might be very inexpensive to run automated tests, but unless they reveal defects of consequence, they may curtail your investment in more expensive manual tests and more rarely run tests that might reveal more defects of consequence.

3) Usefulness of automation. Do automated tests have continuing value to either find latent bugs or to show regressions? At the system level, automated tests are usually targeted around regression testing scenarios and non-functional qualities such as performance or security.

So far the rules of thumb for when to automate and when not to automate boil down to:

- **Unit tests** are typically **automated** – their automation cost is typically a low multiple of their manual test cost, they are run frequently while a product is evolving, their maintenance cost is usually not more than the cost of changing the code itself.
- **Build verification tests** are typically **automated** – their automation cost is typically outweighed by the number of times they will be repeated, the consistency of information desired (same exact values), and moderate maintenance costs.
- **Usability tests** typically use **manual exploration** – they are nearly impossible to automate, not typically affected by every software change and thus have no need to repeat as frequently, and have potentially very high maintenance costs.

5.1 Application Programming Interface (API) Testing

Most API level tests are typically automated, since code must be written to invoke the APIs. You can create interactive shells that allow calling APIs from an interactive scripting-like environment, but they sometimes limit you versus the native implementation. You can also automatically record the exploration and nearly automatically turn it into an automated test script. Capture/Replay for APIs is more reliable than for User Interfaces (UIs, such as command line or graphical). Exploring APIs in a non-automated fashion is a way to verify usability of an API, but so is building software similar to the anticipated usage of the APIs using scenarios. Finally, APIs easily lend themselves to many of the simpler analysis techniques so that creating high coverage automated tests should not be much more expensive than manually testing APIs.

In any testing, you must understand the nature of what you are testing and how deterministic it is. Non-deterministic results may make automatic testing difficult regardless of API or UI. Non-deterministic outputs from machine learning and artificial intelligence programs are particularly difficult. This includes grading results from a Search or a complex distributed system.

A simple example I experienced was a set of SQL tests. The SQL Language does not guarantee that the results of 3 inserts into a database will return those 3 items in the order inserted. However, testers naively observed and assumed that. When the underlying SQL engine was changed into a parallelized version, most of the tests broke because they made assumptions beyond those allowed by the APIs. Writing tests to not assume more than the API can be very difficult at times. Note, that it doesn't get any easier for manual testing humans either as now noticing whether 1 record out of a 1000 is missing is quite error prone. Most humans are not good at detail as evidenced by typographical errors overlooked by humans every day.

5.2 Performance and Stress testing

If you want to monitor performance over time to check for degradation or improvement, then you want repeatable performance tests. The more often you want to repeat them, then the more likely automated tests will be applicable. If you are verifying small, atomic operations like a single API call or Database retrieval, then automation becomes critical because the time scale is too small. Any performance testing that takes under a few seconds would be unreliably performed by humans.

Load and stress tests generally use automation to create the load and stress. However, I have seen reports of large manual load or stress tests. This is especially true when testing a large, worldwide distributed, production system. A typical compromise is to combine automated testing for generating the load and stress with manual exploration concurrently looking for anomalous behavior the automation can easily miss.

5.3 Software Version stages

Beyond the types listed above, there are still numerous areas for consideration. Some questions to consider:

Should integration and system tests be automated?
Where are your company and its product in its lifecycle?

The ROI for automation frequently occurs after the first release. If just getting a quality first release is the biggest concern, as for example in a startup, then defect finding at the system level via manual testing may be of more benefit than automation to save future costs. When a company has an expectation to maintain the software for 10 years and even if developing the first version of a product, then future cost reduction via more spending now on automation of system testing may be of most benefit. A company may choose to ship fewer features in order to fully automate the ones shipped. Another frequent approach is loosely the following for version V of a product:

Assume mostly automated tests for the previous version, say V-1.
As version V is developed, new features are manually tested, old features are automatically regression tested using the V-1 test cases.
When version V is shipped, as planning is beginning for V+1, the V test cases are automated.

There are many dangers to the above approach:

1) If planning for V+1 occurs during the end of the version V cycle, then there is never time for test automation.
2) If test is automating version V tests at the start of the V+1 cycle, then test may not have time to participate in the planning (and Marick questions whether they should, Marick99).

So lifetime of the product will affect the anticipated number of reruns of the test cases, but other factors can still affect when they should be automated. Perhaps you don't automate the test cases until after the product being tested proves to be profitable.

6 Partial automation – S.E.A.R.C.H.

Automation is not a binary question. Good tests typically have six parts as described with the acronym SEARCH in Bergman92. SEARCH stands for **S**etup, **E**xecution, **A**nalysis, **R**eporting, **C**leanup, and **H**elp.

Setup	provides the assumed environment (either by verifying it exists or creating the environment).
Execution	of the test is running the software. Code coverage tools measure what was executed, not what was verified.
Analysis	is the verification. It may be asserts in the test code as it ran or it can be post processing of whether the software did what it was supposed to do and didn't have any unintended side effects on the system as a whole.
Reporting	the results of the analysis can include roll ups and statistics.
Cleanup	is the final phase that returns the software to a known state so that the next test can proceed.
Help	is the associated documentation for the tests: how they were designed to operate and how to keep them runnable and maintainable. This is particularly true regarding which tests can be run in parallel and which must be run separately.

Each of these can be done in an automated or manual fashion for justifiable reasons. I've seen testers automate their execution time down to 10 minutes while repeatedly doing manual setups of one hour. They should have focused their attention on Setup.

6.1 Examples of Automated and Manual analysis

When tests are repeated in many environments, then analysis of failures (not their detection), may become a predominant cost, so automated failure analysis may be the most useful automation.

Automated Failure Analysis is a means of determining if a specific observed failure has previously been diagnosed. See Travison08 and Staenff07 whose subtitle: “Why Weak Automation Is Worse Than No Automation” really strikes home.

Another example is verifying UI output. Frequently minor variances in UI can be critical or inconsequential. Computers have a difficult time distinguishing. Computers can easily distinguish if something is different, but not the significance of the difference. So one example of automation is to automatically record differences (analysis) and later provide humans the ability to manually evaluate their significance (reporting).

7 Conclusion

How are you repeating your test? Manually via humans or automated via software?
Software test automation is merely a method of repetition. When to automate software testing is dependent on many factors mostly around repetition.

When are you repeating your tests? How often? Every input value or combination of inputs? Every day or once a year? The more you repeat the more likely automation makes sense.

Why are you repeating your tests? Change to the inputs, environment, or software? Changing the inputs requires an understanding of what the expected outputs will be. For UI testing, a human can sometimes more quickly and easily judge if the expected output looks right. For changing environment and software you are usually expecting the same output, and automated tests are typically more reliable at detecting changes.

What are you repeating about your tests? What are you varying? Repeating the same inputs makes automation more likely. Varying an environment may be expensive and difficult and thus only manually done. Repeatedly using the same SUT means you are looking for latent defects, not regressions, and exploratory testing may be most valuable.

8 References

Bergman M., and K. Stobie, 1992 *How to Automate Testing-the Big Picture*, Quality Week 1992
http://keithstobie.net/Documents/TestAuto_The_BigPict.pdf

Dustin, E., et al 1999, *Automated Software Testing: Introduction, Management, and Performance*, Addison-Wesley Professional 1999. ISBN: 978-0201432879

Gamma, E., et al 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994 ISBN: 978-0201633610

Hoffman, D. 1999, *Cost Benefits Analysis of Test Automation*, Proceedings of the Software Testing Analysis and Review Conference (STAR East). Orland, Florida. May 1999
http://www.softwarequalitymethods.com/Papers/Star99%20model%20Paper.pdf

Kaner, C., J. Bach and B. Pettichord. 2001, *Lessons Learned in Software Testing*, Wiley 2001

Marick, B. 2000, *Testing For Programmers*, (pages 43-47)
http://www.exampler.com/testing-com/writings/half-day-programmer.pdf

Marick, B. 1999, *Maybe Testers Shouldn’t Be Involved Quite So Early*
http://www.exampler.com/testing-com/writings/testers-upstream.pdf

Marick, B. 1998, *When Should a Test Be Automated?*, Quality Week 1998
http://www.exampler.com/testing-com/writings/automate.pdf

Mosley, D. and B. Posey 2002, *Just Enough Software Test Automation*, Prentice Hall PTR, 2002.
ISBN: 978-0130084682

Saran, c 2008, *Autistic people prove valuable in software testing*, ComputerWorld, 15 Feb 2009,
http://www.computerweekly.com/Articles/2008/02/15/229432/autistic-people-prove-valuable-in-software-testing.htm

Staneff, G., 2007, *Test Logging and Automated Failure Analysis*, SASQAG September 2007.
http://www.sasqag.org/pastmeetings/9-2007/SASQAG_9-20-2007.ppt

Travison, D. and Staneff, G. 2008, *Test Instrumentation and Pattern Matching for Automatic Failure Identification*, 1st International Conference on Software Testing, Verification, and Validation, 2008
ISBN: 978-0-7695-3127-4

Whittaker, J. 2002, *How to Break Software: A Practical Guide to Testing*, Addison Wesley 2002
ISBN: 978-0201796193