DEDICATED TO SHOWCASING NEW TECHNOLOGY AND WORLD LEADERS IN SOFTWARE TESTING

LogiGear MAGAZINE

November Issue - Volume 4

TESTING IN AGILE PART 5: TOOLS

ARE USE CASES HARMFUL FOR TEST AUTOMATION?

By Hans Buwalda

Dr. Cem Kaner
Expert Interview

"I've always been fascinated by large collections of data - whether there might be some interesting or useful properties hiding in a mass of numbers."

An interview with D. Richard Kuhn, Computer Scientist, NIST

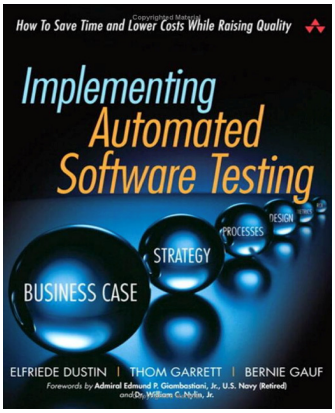
HAPPY THANKSGIVING 2010

www.logigearmagazine.com

IN THIS ISSUE

November Issue – Volume 4

CONTENTS



Spotlight Interview

In Brief

Feature Articles

Related Articles

★ LETTER FROM THE EDITOR

Hi everyone and welcome to our fourth edition of LogiGear Magazine. This month we finish Michael Hackett's piece on "Agile in Testing" with part five, Tools. I think this makes for a nice conclusion to a very exhaustive piece on what testers can expect when they are thrust into an Agile Development project. We also have more exciting interviews from leading professors in computer science; this month we interview Computer Scientist D. Richard Kuhn, who shares with us his thoughts on combinatorial testing methods, very interesting stuff. Hope you continue to enjoy our magazine; we are working hard at trying to make it one of the best in the testing industry. Hope you had a wonderful Thanksgiving and all the best in the coming New Year.

William Coleman

Testing in Agile Part 5: TOOLS

To begin this article, it would be a good idea to, remember this key point:

Agile Manifesto Value #1
Individuals and interactions over processes and tools

Tools work at the service of people. People, particularly intelligent people, can never be slaves to tools. People talking to each other, working together and solving problems is much more important than following a process or conforming to a tool. This also means someone who tells you a tool will solve your Agile problems knows nothing about Agile. A disaster for software development over the past decade has been companies investing gobs of money into "project management tools" or "test management tools" to replace, in some cases, successful practices of software development, in favor of the process the tool or tool vendor told that company was "best practice". That made certain tool vendors rich and many software development teams unhappy.

If your team wants to be Agile, and work in line with the value outlined above, I have a couple of rules to remember when it comes to tools and Agile.

Rule #1 - Tools need to fit teams. Teams should not fit into tools!
Rule #2 - Test teams need to grow in their technical responsibility.

These two ideas sum up how I feel about most tools that are part of an Agile development project. For example, when you add management tools, this will add overhead, reduce efficiency and limit the Agile idea of self-directing teams. However, communication tools can help in distributed development situations and scaling into large organizations. Some tools work to provide services to teams and others do not. This article is mainly about tools and how they relate to test teams. We are going to talk a lot about the different toolsets- that are most important to test teams. We won't talk about unit test frameworks since test teams very rarely manage them, but we will discuss test case managers in the Agile world.
I'm also going to talk about the changes that the test teams need to undergo to successfully implement or use these new tools.

Often when beginning an Agile project, there is a need for some re-tooling in order to be successful. The same set of tools we used in traditional development may not be enough to support rapid communication, dynamic user story change and ranking, and increased code development.

The groups of tools of specific use to test teams we will focus on are:

- Application Lifecycle Management (ALM) – such as Rally, Atlassian's JIRA Studio and IBM's Rational Team Concept
- Test Case Management Tools – such as OTHER**, and HP's Quality Center
- Continuous Integration Tools – such as CruiseControl, Bamboo, and Hudson (Including Source control tools – such as PerForce, and Visual Source Safe (VSS) type tools)
- Automation Tools – we will not talk about specific tools in this section.

Application Lifecycle Management Tools (ALM)

Sometimes when we talk about tools, we discuss them in relation to the Application Lifecycle. Naturally then, the tool vendors refer to their large suites as Application Lifecycle Management Tools, or ALM. This is the major category of tooling and it is often an enormous undertaking to get these tools implemented. However, there are some minor category tools in the ALM spectrum that we can pull out that relate to testers.

Application Lifecycle Management (ALM) tools have been making a splash in the development world for a few years now. Approximately 15 years ago, Rational (before Rational/IBM) had an entire suite of tools we would now call ALM. From RequisitePro to Rose

to ClearCase through ClearQuest, the idea was to have one set of integrated tools to manage software development projects from inception to deployment. Now, there are multitudes of tool sets in this space. The use of these tools has grown in the past couple of years as teams have became more distributed, faster and more dynamic while at the same time coming under increased pressure from executives to be more measurable and manageable. When Agile as a development framework grew explosively, the anti-tool development framework got inundated by tools.

ALM tool suites generally manage user stories/requirements, call center/support tickets, also help track planning poker, store test cases associated with user stories, bugs – there are many tools that do this - but the big leap forward with current ALM tools is them being linked to source control, unit testing frameworks, and GUI test automation tools. Lite ALM tools satisfy the need for speed required by the rapid nature of software development – it's a communication tool.

There are *great reasons for using ALM tools*, just as there are great reasons not to use them! Let's start with great reasons to use them:

1. If you're working on products with many interfaces to other teams and products, ALM tools can be very beneficial in communicating what your team is doing and where you are in the project.

2. ALM tools are essential if you're working with distributed teams.

3. If you are working on projects with large product backlogs, or with dynamic backlogs often changing value, a management tool can be very helpful.

4. ALM tools are useful if you have teams offshore or separated by big time zone differences

5. You are scaling Agile - for highly integrated teams, enterprise communication tools may be essential.

6. If, for whatever reason, the test team is still required to produce too many artifacts- mainly the high overhead artifacts like test cases and bug reports, build reports, automation result report, and status reports (ugh!) – most ALM tools will have all the information you need to report readily available.

There are many good reasons to use ALM tools. But the tool can't control you or the use and benefit of going Agile will be lost.

Most Agile lecturers, including Ken Schwaber, will tell you using Agile in distributed teams will reduce the efficiency. This does not mean Agile cannot be successful with offshoring, but it means it is tougher and chances of ending up with smiles all around are lower. Nothing beats all being in the same place, communicating easily and openly every day and working on our software in my bullpen in the office. If you are in that situation - good for you! However, this is the subject of another magazine article we will bring you in the New Year.

Now the *great reasons **not** to use ALM tools*:

1. It is very easy for some people to become tool-centric, as though the tool is the project driver rather than the Teams. That is completely *waterfall*, bad!
2. Paper trails and the production of artifacts, "because we always did it this way," are anti-Agile. Most times the artifacts become stale or useless having little relevance to what is actually built.
3. Management overheard is anti-Agile. The only "project management" if I can even use that phrase, is a 15 minute daily scrum. Not a giant, administrative nightmare, high cost overhead, project management process! When the ALM tool becomes a burden and overhead slows down the team, you will need to find ways to have the tools support you rather than slow you down.

A quick word about measurement: the only artifact about measurement in line with *by-the-book* Agile is burndown charts. Extreme

programming espouses velocity. If your team is measuring you by test case creation, test case execution rates, user story coverage there is nothing Agile about it. End of discussion.

If you want to investigate ALM tools further, I recommend you look at a few - check out enterprise social software maker, *Atlassian*, maker of the famous Jira, and *Rally* a great, widely used product with dozens of partners for easy integration. These two are at the opposite ends of the cost spectrum.

Test Case Management Tools

Most test organizations have migrated, over the past decade, to using an enterprise wide test case management tool (TCM) for tracking test cases, bugs/issues and support tickets in the same repository. Many test teams have strongly mixed feeling about these tools- sometimes feeling chained to them as they inhibit spontaneous thinking and technical creativity.

To begin talking about test case tools in Agile, there are two important reminders:

1. Test cases are *not* a prescribed or common artifact in Scrum.
2. Most companies that are more Agile these days are following some lean manufacturing practices - such as ceasing to write giant requirement docs and engineering specs, and getting really lean on project documentation. Test teams need to get lean also. They can start by cutting back or cutting out test plans and test cases. Do you need a tool to manage test cases in Agile? Strict process says no!

If you are not using a TCM yet and are moving to Agile, or are beginning to use a test case manager, now is not a good idea If you already use one - get leaner!
Still, most ALM tools have large functionality built around test case management that can become an overhead problem for teams focusing on fast, lean, streamlined productivity.

Continuous Integration

In rapid development, getting and validating a new build can kill progress. Yet there is no need for this! In the most sophisticated and successful implementations of Agile development, test teams have taken over source control and the build process. We have already talked about continuous integration in this series (See *Agile For Testers Part 3 Automated Build and Build Validation Practice / Continuous Integration* for more discussion on this), let's focus on tools. Source control tools are straightforward. Many test teams regularly use the source controls systems.

There's no reason why a test team can't take over making a build. There is also no reason a test team does not have someone already technically skilled enough to manage continuous integration. With straightforward continuous integration tools such as Bamboo, CruiseControl and Build Forge, test teams can do it.
My advice is:

- Learn how these tools work and what they can do
- Build an automated smoke test for build verification
- Schedule builds for when you want delivery

Test teams can use a continuous integration tool to make a big build validation process: scheduling new builds, automatic re-running of the unit tests, re-running automated user story acceptance tests and whatever automated smoke tests are built! Note, this does not mean write the unit tests - it means re-run, for example, the J-unit harness of tests. Test teams taking over the continuous integration process is, from my experience, what separates consistently successful teams from teams that enjoy only occasional success. Easy.

I suggest you visit Wikipedia on CI tools here: http://en.wikipedia.org/wiki/Continuous_integration) to see a very large list of continuous integration tools for various platforms.

Test Automation

I'm going to resist going into detail about automation here. Automation in Agile will have its own large magazine articles and white papers in 2011. Remember, when discussing automation in Agile with team members, scrum masters or anyone knowledgeable in Agile, the part of testing that a test team needs to automate is not the unit testing. That automation comes from developers. Test teams need to focus on:

- Automated smoke/ build validation tests
- Large automated regression suites
- Automated user story validation tests (typically a test team handles this. But- there are very successful groups that have developers automate these tests at the unit, api or UI level)
- New functionality testing can be difficult to automate during a sprint. You need a very dynamic and easy method for building test cases. This will be discussed a greater length in an future Agile automation methods paper.

In Agile development, the automation tool is not the issue or problem, it's *how* you design, describe and define your tests that will make them Agile or not!

Here are the main points to get about test automation tools in Agile:

- The need for massive automation is essential, clear and non-negotiable. How test teams can do that in such fast-paced development environments needs to be discussed fully in its own white paper. That you need a more manageable and maintainable, extendible framework, like tools that easily support action-based testing is clear.
- Developers need to be involved in test automation! Designing for testability and designing for test automation will solve automation problems.
- Training is key to building a successful automation framework.
- Tools never solve automation problems- free or otherwise! Better automation design solves automation problems.

Summary:

Tools in Agile need to be used to fit people, not the other way around.

Tools need to be easy to use (from your perspective- not the tool vendor's perspective!) and low time and management overheads.

Employ lean manufacturing ideas wherever possible- specifically, cut artifacts and documentation that is not essential.

What is Combinatorial Testing and Why Should Testers Care?

By Rick Kuhn, Raghu Kacker and Yu Lei

Developers of large data-intensive software often notice an interesting—though not surprising—phenomenon: When usage of an application jumps dramatically, components that have operated for months without trouble suddenly develop previously undetected errors. For example, the application may have been installed on a different OS-hardware-DBMS-networking platform, or newly added customers may have account records with an oddball combination of values that have not occurred before. Some of these rare combinations trigger failures that have escaped previous testing and extensive use. Such failures are known as *interaction failures*, because they are only exposed when two or more input values interact to cause the program to reach an incorrect result.

Combinatorial testing can help detect problems like this early in the testing life cycle. The key insight underlying this method is that not every parameter contributes to every failure and most failures are triggered by a single parameter value or interactions between a relatively small number of parameters. To detect interaction failures, software developers often use "pairwise testing", in which all possible pairs of parameter values are covered by at least one test. Its effectiveness is based on the observation that software failures often involve interactions between parameters. For example, a router may be observed to fail only for a particular protocol when packet volume exceeds a certain rate, a 2-way interaction between protocol type and packet rate. Figure 1 illustrates how such a 2-way interaction may happen in code. Note that the failure will only be triggered when both *pressure* < *10* and *volume* > *300* are true.

```
if (pressure < 10) {
      // do something
      if (volume > 300)
{
            faulty code!
      BOOM!
      }
      else {
            good code, no
      problem
      }
}
else {
```

Figure 1. 2-way interaction failure triggered only when two conditions are true.

But what if some failure is triggered only by a very unusual combination of 3, 4, or more sensor values? It is very unlikely that pairwise tests would detect this unusual case; we would need to test 3-way and 4-way combinations of values. But is testing all 4-way combinations enough to detect all errors? What degree of interaction occurs in real failures in real systems? Surprisingly, this question had not been studied when NIST began investigating interaction failures in 1999. Results showed that across a variety of domains, all failures could be triggered by a maximum of 4-way to 6-way interactions. As shown in Figure 2, the detection rate (*y* axis) increased rapidly with interaction strength (the interaction level *t* in *t*-way combinations is often referred to as *strength*). Studies by other researchers have been consistent with these results.

Failures appear to be caused by interactions of only a few variables, so tests that cover all such few-variable interactions can be very effective.

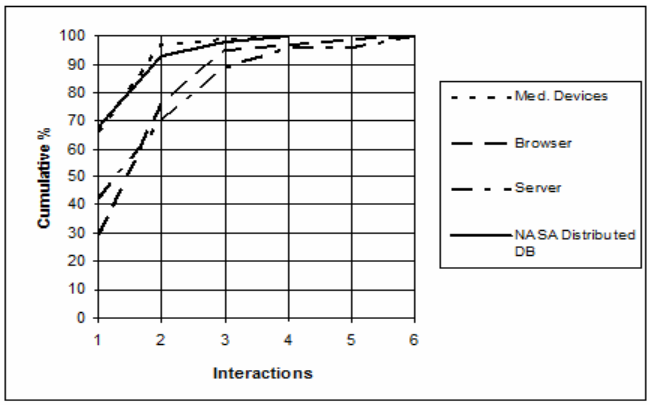


Figure 2. Error detection rates for interaction strengths 1 to 6

These results are interesting because they suggest that, while pairwise testing is not sufficient, the degree of interaction involved in failures is relatively low. Testing all 4-way to 6-way combinations may therefore provide reasonably high assurance. As with most issues in software, however, the situation is not that simple. Most parameters are continuous variables which have possible values in a very large range (+/- 2^{32} or more). These values must be discretized to a few distinct values. In addition, we need to determine the correct result that should be expected from the system under test for each set of test inputs. But these challenges are common to all types of software testing, and a variety of good techniques have been developed for dealing with them.

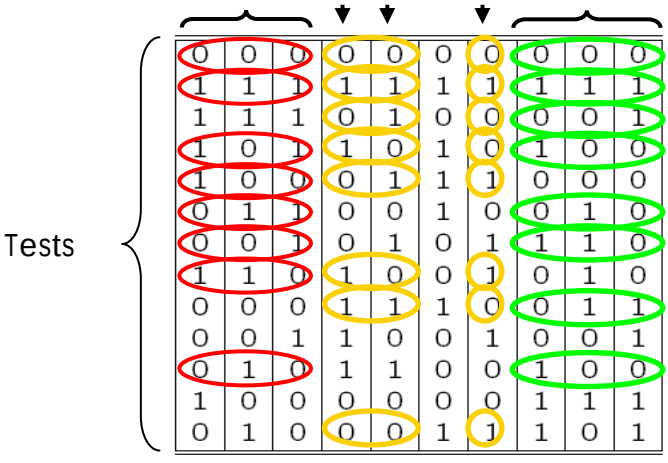


Figure 3. 3-way covering array

A test suite to cover all 2-way, 3-way, etc. combinations is known as a *covering array*. An example is given in 0, which shows a 3-way covering array for 10 variables with two values each. The interesting property of this array is that any three columns contain all eight possible values for three binary variables. For example, taking columns F, G, and H, we can see that all eight possible 3-way combinations (s000, 001, 010, 011, 100, 101, 110, 111) occur somewhere in the three columns together. In fact, any combination of three columns chosen in any order will also contain all eight possible values. Collectively, therefore, this set of tests will exercise all 3-way combinations of input values in only 13 tests, as compared with 1,024 for exhaustive coverage.

How can Combinatorial Methods be Used in Testing?

There are basically two approaches to combinatorial testing – use combinations of *configuration* parameter values, or combinations of *input* parameter values. In the first case, we select combinations of values of configurable parameters. For example, telecommunications software may be configured to work with different types of call (local, long distance, international), billing (caller, phone card, 800), access (ISDN, VOIP, PBX), and server for billing (Windows Server, Linux/MySQL, Oracle).

In the second approach, we select combinations of *input data* values, which then become part of complete test cases, creating a test suite for the application. For example, a word processing application may allow the user to select 10 ways to modify some highlighted text: *subscript, superscript, underline, bold, italic, strikethrough, emboss, shadow, small caps*, or *all caps*. Thorough testing requires that the font-processing function work correctly for all valid combinations of these input settings. But with 10 binary inputs, there are 2^{10} = 1,024 possible combinations. But the empirical analysis reported above shows that testing all 3-way combinations may detect 90% or more of bugs. For a word processing application, testing that detects better than 90% of bugs may be a cost-effective choice. The covering array shown in Figure 3 could be used for this testing.

Conclusions

Using combinatorial methods for either configuration or input parameter testing can help make testing more effective at an overall lower cost. Although these methods do not apply to all applications, they can be a valuable addition to the tester's toolbox.

Certain commercial products are identified in this document, but such identification does not imply recommendation by the US National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.

(This article is condensed from **Chapter 2 of Practical Combinatorial Testing**, freely downloadable here)

Rick Kuhn is a computer scientist at the Computer Security Division of the US National Institute of Standards and Technology (NIST).

Raghu Kacker is a mathematical statistician at the Mathematical and Computational Sciences Division of NIST.

Yu Lei is an associate professor at the Department of Computer Science and Engineering at the University of Texas at Arlington.

WHAT IS COMBINATORIAL TESTING?

All-pairs testing or **pairwise testing** is a combinatorial software testing method that, for each pair of input parameters to a system (typically, a software algorithm), tests all possible discrete combinations of those parameters. Using carefully chosen test vectors, this can be done much faster than an exhaustive search of all combinations of all parameters, by "parallelizing" the tests of parameter pairs. The number of tests is typically O(*nm*), where *n* and *m* are the number of possibilities for each of the two parameters with the most choices. (Wikipedia, 2010)

Combinatorial Software Testing

By *D. Richard Kuhn, Computer Scientist, NIST*
Raghu Kacker, Mathematical Statistician, NIST
Yu Lei, Associate Professor, University of Texas at Arlington
Justin Hunter, CEO, Hexawise

"Combinatorial testing can detect hard-to-find software faults more efficiently than manual test case selection methods."

Developers of large data-intensive software often notice an interesting—though not surprising—phenomenon: When usage of an application jumps dramatically, components that have operated for months without trouble suddenly develop previously undetected errors. For example, newly added customers may have account records with an oddball combination of values that have not been seen before. Some of these rare combinations trigger faults that have escaped previous testing and

extensive use. Alternatively, the application may have been installed on a different OS-hardware-DBMS-networking platform.

Combinatorial testing can help detect problems like this early in the testing life cycle. The key insight underlying t-way combinatorial testing is that not every parameter contributes to every fault and many faults are caused by interactions between a relatively small number of parameters.

Table 1. Pairwise test configurations.

Test case	OS	CPU	Protocol
1	Windows	Intel	IPv4
2	Windows	AMD	IPv6
3	Linux	Intel	IPv6
4	Linux	AMD	IPv4

PAIRWISE TESTING

Suppose we want to demonstrate that a new software application works correctly on PCs that use the Windows or Linux operating systems, Intel or AMD processors, and the IPv4 or IPv6 protocols. This is a total of 2 × 2 × 2 = 8 possibilities but, as Table 1 shows, only four tests are required to test every component interacting with every other component at least once. In this most basic combinatorial method, known as pairwise testing, at least one of the four tests covers all possible pairs (t = 2) of values among the three parameters.

Note that while the set of four test cases tests for all pairs of possible values—for example, OS = Linux and protocol = IPv4—several combinations of three specific values are not tested—for example, OS = Windows, CPU = Intel, and protocol = IPv6.

Even though pairwise testing is not exhaustive, it is useful because it can check for simple, potentially problematic interactions with relatively few tests. The reduction in test set size from eight to four shown in Table 1 is not that impressive, but consider a larger example: a manufacturing automation system that has 20 controls, each with 10 possible settings — a total of 1020 combinations, which is far more than a software tester would be able to test in a lifetime. Surprisingly, we can check all pairs of these values with only 180 tests if they are carefully constructed.

Figure 1 shows the results of a 10-project empirical study conducted recently by Justin Hunter that compared the effectiveness of pairwise testing with manual test case selection methods.

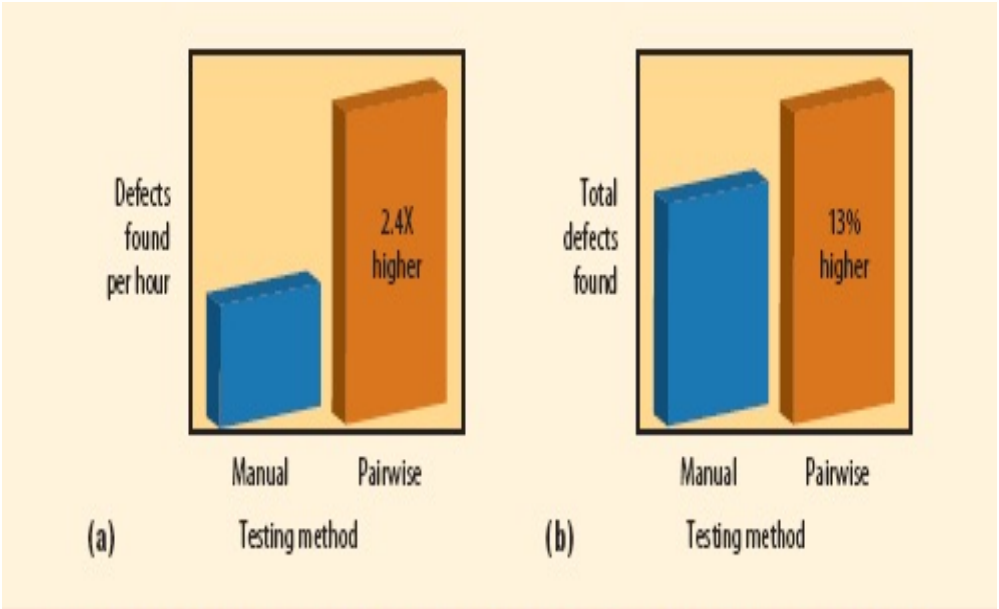


Figure 1. Summary of results from 10 projects. Pairwise combinatorial test case selection versus manual test case selection: (a) testing efficiency and (b) testing quality.

The projects were conducted at six companies and tested commercial applications in development; in each project, two small teams of testers were asked to test the same application at the same time using different methods. One group of testers selected tests manually; they relied on "business as usual" methods such as developing tests based on functional and technical requirements and potential use cases mapped out on whiteboards. The other group used a combinatorial testing tool to identify pairwise tests. Test execution productivity was significantly higher in all of the projects for the testers using combinatorial methods, with test execution productivity more than doubling on average and more than tripling in three projects. The groups using pairwise testing also achieved the same or higher quality in all 10 projects; all of the defects identified by the teams using manual test case selection methods were identified by the teams using combinatorial methods. In five projects, the combinatorial teams found additional defects that had not been identified by the teams using manual methods.

These proof-of-concept projects successfully demonstrated to the teams involved that manual methods of test case selection were not nearly as effective as pairwise combinatorial methods for finding the largest number of defects in the least amount of time.

TESTING HIGHER-DEGREE INTERACTIONS

Other empirical investigations have concluded that from 50 to 97 percent of software faults could be identified by pairwise combinatorial testing. However, what about the remaining faults? How many failures could be triggered only by an unusual interaction involving more than two parameters?

In a 1999 study of faults arising from rare conditions, the National Institute of Standards and Technology reviewed 15 years of medical device recall data to determine what types of testing could detect the

reported faults (D.R. Wallace and D.R. Kuhn, "Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data," Int'l J. Reliability, Quality, and Safety Eng., Dec. 2001, pp. 351-371). The study found one case in which an error involved a four-way interaction among parameter values: demand dose = administered, days elapsed = 31, pump time = unchanged, and battery status = charged.

Pairwise combinatorial testing is unlikely to detect faults like this because it only guarantees that all pairs of parameter values will be tested. A particular four-way combination of values is statistically unlikely to occur in a test set that only ensures two-way combination coverage; to ensure thorough testing of complex applications, it is necessary to generate test suites for four-way or higher-degree interactions.

Investigations of other applications found similar distributions of fault-triggering conditions. Many faults were caused by a single parameter, a smaller proportion resulted from an interaction between two parameter values, and progressively fewer were triggered by three-, four-, five,- and six-way interactions. **Figure 2** summarizes these results. Thus far, a fault triggered by a seven-way interaction has not appeared.

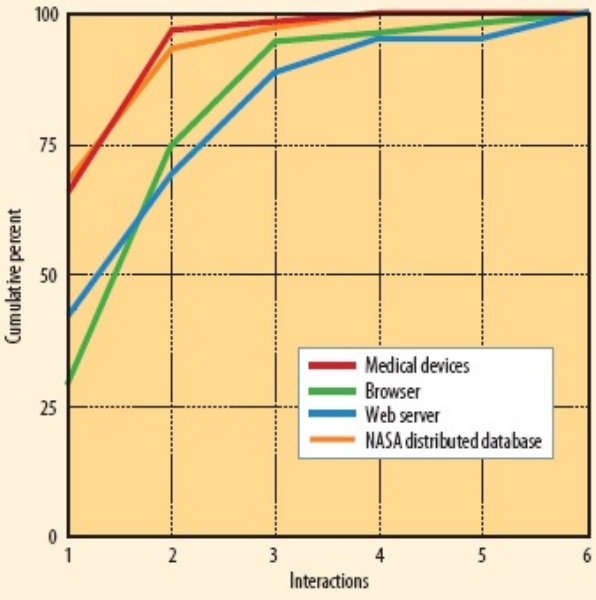
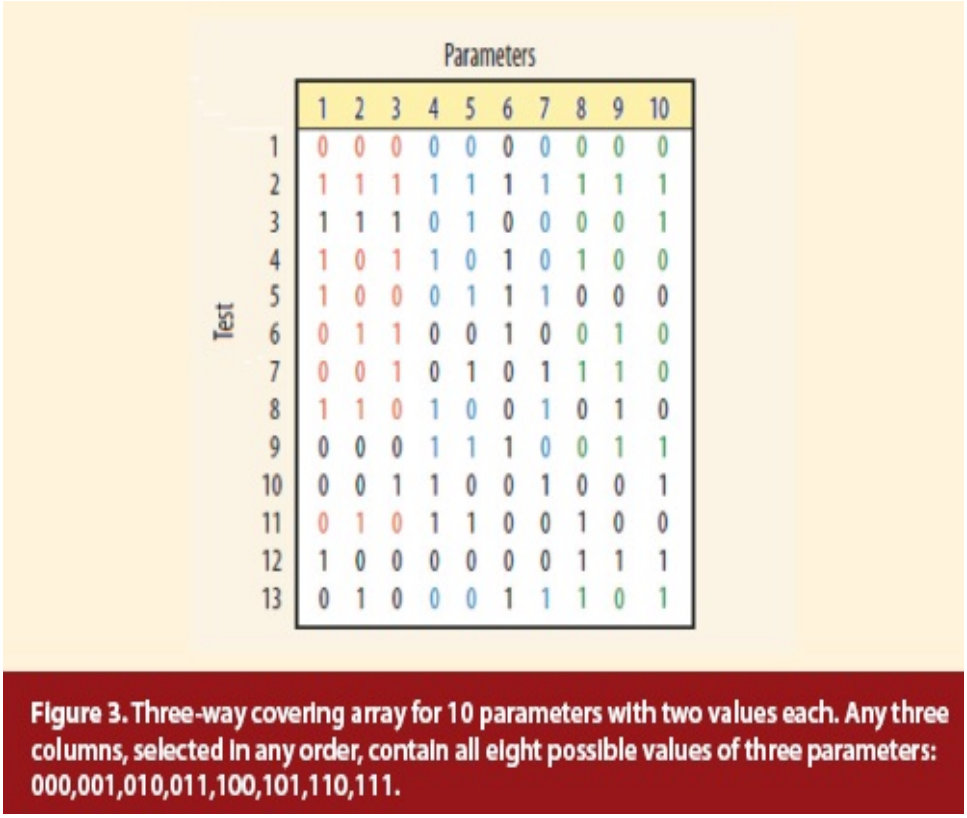


Figure 2. Cumulative error detection rate for fault-triggering conditions. Many faults were caused by a single parameter value, a smaller proportion resulted from an interaction between two parameter values, and progressively fewer were triggered by three-, four-, five, and six -way interactions.

With the Web server application, for example, roughly 40 percent of the failures were caused by a single value, such as a file name exceeding a certain length; another 30 percent were triggered by the interaction of two parameters; and a cumulative total of almost 90 percent were triggered by three or fewer parameters. While not conclusive, these results suggest that combinatorial methods can achieve a high level of thoroughness in software testing.

The key ingredient for this kind of testing is a covering array, a mathematical object that covers all t-way combinations of parameter values at least once. For the pairwise testing example in Table 1, t = 2, and it is relatively easy to generate tests that cover all pairs of parameter values. Generating covering arrays for complex interactions is much harder, but new algorithms make it possible to generate covering arrays orders of magnitude faster than previous algorithms, making up to six-way covering arrays tractable for many applications.

Figure 3 shows a covering array for all three-way interactions of 10 binary parameters in only 13 tests. Note that any three columns, selected in any order, contain all eight possible values of three parameters: 000,001,010,011, 100,101,110,111. Three-way interaction testing detected roughly 90 percent of bugs in all four of the empirical studies in Figure 2, but exhaustive testing of all possible combinations in **Figure 3** would require 210 = 1,024 tests.



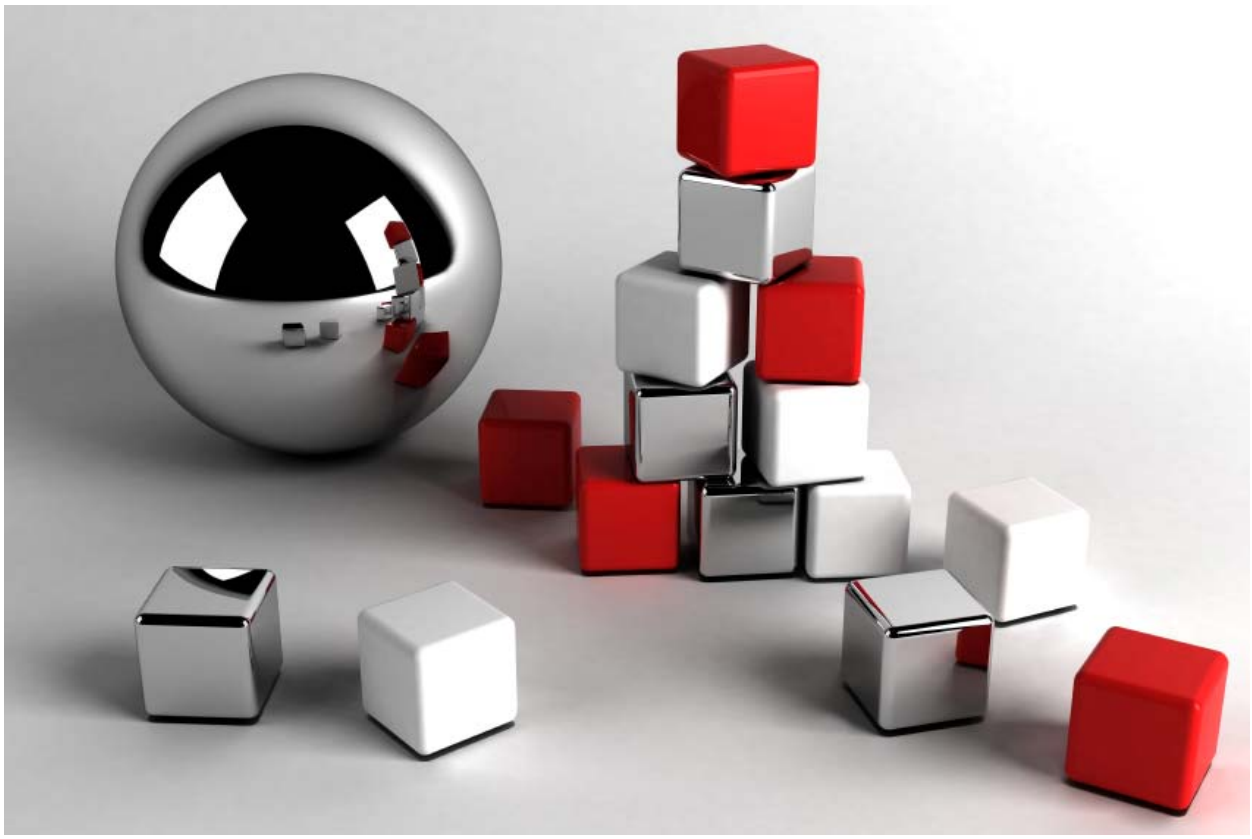
Test	Parameters									
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	0	1	0	0	0	0	1
4	1	0	1	1	0	1	0	1	0	0
5	1	0	0	0	1	1	1	0	0	0
6	0	1	1	0	0	1	0	0	1	0
7	0	0	1	0	1	0	1	1	1	0
8	1	1	0	1	0	0	1	0	1	0
9	0	0	0	1	1	1	0	0	1	1
10	0	0	1	1	0	0	1	0	0	1
11	0	1	0	1	1	0	0	1	0	0
12	1	0	0	0	0	0	0	1	1	1
13	0	1	0	0	0	1	1	1	0	1

Figure 3. Three-way covering array for 10 parameters with two values each. Any three columns, selected in any order, contain all eight possible values of three parameters: 000,001,010,011,100,101,110,111.

What are the pragmatic implications of being able to achieve 100 percent three-way coverage in 13 test cases on real-world software testing projects? Assuming that there are 10 defects in this hypothetical application and that 9 are identified through the 13 tests indicated, testing these 13 cases would find 71 times more sdefects per test case [(9/13)/(10/1,024)] than testing exhaustively and uncovering all 10.

While the most basic form of combinatorial testing — pairwise — is well established, and adoption by software testing practitioners continues to increase, industry usage of these methods remains patchy at best. However, the additional training required is well worth the effort. Teams seeking to maximize testing thoroughness given tight time or resource constraints, and which currently rely on manual test case selection methods, should consider pairwise testing. When more time is available or more thorough testing is required, t-way testing for t > 2 is better. Practitioners who require very high quality software will find that covering arrays for higher-strength combinations can detect many hard to-find faults, and variability among detection rates appears to decrease as t increases.

Sophisticated new combinatorial testing algorithms packaged in user-friendly tools are now available to enable thorough testing with a manageable number of test cases and at lower cost, and make it practical for testers to develop empirical results on applications of this promising test method.



Allpairs, Pairwise, Combinatorial Analysis

By BJ Rollison, Test Architect, Microsoft

Last week I went to StarWest as a presenter and as a track chair to introduce speakers. Being a track chair is wonderful because you get to interface more closely with other speakers. Anyway, one of the speakers I introduced was Jon Bach. Jon is a good public speaker, and I was pleasantly surprised that he was doing a talk on the allpairs testing technique (also known as pairwise or combinatorial analysis). I wish Jon dedicated a little more time to the specifics of the technique during his talk and was generally more aware of available tools and information for folks to investigate further, but I think he successfully raised the general awareness and interest in pariwise testing as an effective testing technique among the audience.

Pairwise testing is one approach to solving the potential explosion in the number of tests when

dealing with multiple parameters whose variables are semi-coupled or have some dependency on variable states of other parameters. For example, in the font dialog of MS Word there are 11 checkboxes for various effects such as superscript, strikethrough, emboss, etc. Obviously these effects have an impact on how the characters in a particular font are displayed and can be used in multiple combinations such as Strikethrough + Subscript + Emboss. The total number of combinations of effects is the Cartesian product of the variables for each parameter, or 2^{11} or 2048 in this example. This doesn't include different font types, styles, etc. which are also interdependent. So, you can see how the number of combinations increases rapidly especially as additional dependent parameters are included in the matrix.

The good news is the industry has a lot of evidence to suggest that most software defects occur from simple interactions between the variables of 2 parameters. So, from a risk based perspective where it may not be feasible to test all possible combinations, how do we choose the combinations out of all the possibilities? Two common approaches include orthogonal arrays and combinatorial analysis.

But, true orthogonal arrays require that the number of variables is the same for all parameters. (Rarely true in software.) It is possible to create "mixed orthogonal arrays" where some combinations of variables will be tested more than once. For example, if we have 5 parameters and one parameter has 5 variables and the remaining 4 parameters only have 3 variables each, we can see from the orthogonal array selector (available on FreeQuality website) the size of the orthogonal array is L_{25} (which basically means the test case will require 25 tests which is still significantly less than the total number of combinations of 405).

The other approach is combinatorial analysis (often referred to as pairwise or allpairs testing) because the approach most commonly used is to use a mathematical formula to reduce the total number of combinations in such a way that each variable for each parameter is tested with each variable from the other parameters at least once. In the above example, the number of tests would be reduced to 16. (Note: some tools will give slightly different results.) However, some tools (such as Microsoft's PICT) also allow for more complex analysis of variable combinations such as triplets and n-wise coverage.

One problem that is hopefully not overlooked by testers using these tools is that some combinations of variables are simply not possible. For example, in the Effects group of the Font dialog it is impossible to check the Superscript checkbox and the Subscript checkbox simultaneously. Therefore, the tester either has to manually modify the output, or use a tool that allows constraints. Again, this is another situation where Microsoft's free tool PICT excels. PICT uses a simple basic-like language for conditional and unconditional constraining of combinations of variables. PICT also allows weighting variables, seeding, output randomization, and negative testing.

I didn't want this to be a PICT sales job, but alas my bias has influenced this post. So, I will conclude by pointing the readers to the Pairwise Testing website. My colleague Jacek Czerwonka has pulled together great resources on the technique of combinatorial analysis including a list of free and commercially available tools, and white papers supporting the value and practicality of this testing technique.

BJ Rollison, Test Architect at Microsoft

BJ Rollison is a Test Architect with Microsoft's Engineering Excellence group where he develops technical training curriculum and teaches Microsoft various testing techniques and methodologies.

BJ started his professional career building custom solutions for small and medium sized businesses for an OEM company in Japan in 1991.

BJ also teaches software testing courses at the University of Washington. He is a frequent speaker at international software testing conferences.

Are Use Cases Harmful For Test Automation?

By Hans Buwalda, CTO, LogiGear Corporation

People who know me and my work probably know my emphasis on good test design for successful test automation. I have written about this in "Key Success Factors for Keyword Driven Testing". In the Action Based Testing (ABT) method that I have pioneered over the years it is an essential element for success. However, agreeing with me in workshops and actually applying the principles in projects turns out quite often to be two different things. Apart from my own possible limitations as a teacher, I see at least one more reason: The way the testing is involved in the development projects.

A typical development project will start with a global understanding of what the system needs to do, which is then detailed out further, for example into use cases. These use cases have proven to be helpful in implementation and various testing efforts, but I'm getting more and more the impression that they may also pose a risk for good test design when they are the only source of information for the test team. There are two reasons:

1. They tend to have a high level of detail

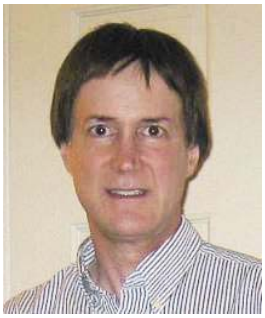
2. They usually follow the end-user perspective

Re 1: The level of detail of use cases is primarily aimed at the developers, and the information they need to know. More often than not it is implicitly assumed that this is also a good level for information for testers to develop test cases from (for the sake of simplicity I will not discuss the usefulness of techniques like exploratory testing, I will just assume that test cases are made and their execution is automated).

Re 2: In many tests it matters how a system handles transactions, and provides the correct and complete follow up actions and information. The end-user interacting with the UI is then not always relevant, and I would not like to see it explicitly specified as part of test cases (in ABT the UI specifics would be hidden in the 'actions'). However, having the UI oriented use cases as the primary source of information makes it hard to focus on the transaction handling and other aspects of the system.

My message would be this: don't start creating test cases from use cases, or similar developer oriented sources of information, before you have had a chance to create a high level test design, in which you specify which test products you're going to create and what level of detail they would need to have.

"A typical development project will start with a global understanding of what the system needs to do, which is then detailed out further, for example into use cases."



Some Insights on Combinatorial Testing

D.Richard Kuhn - Computer Scientist, National Institute of Standards & Technology

LogiGear: How did you become interested in developing applications for combinatorial research? What led you to it personally, and what did you find fascinating about it?

Mr. Kuhn: About 12 years ago Dolores Wallace and I were investigating causes of software failures in medical devices. About that time, Raghu Kacker in our math division introduced me to some work on the design of experiments and testing, which had been done by a colleague of his at Bell Labs. The idea behind these methods was that some failures only occur as a result of an interaction between components. For example, a system may correctly produce error messages when space is exhausted or when input rate exceeds some limit, but crashes only when these two conditions are both true at the same time. Pairwise testing has been used for a long time to catch this sort of problem.

I wondered if we could determine what proportion of the medical device failures were caused by interactions, and just how complex the interactions would be – in other words, how many failures were triggered by just one parameter value, and how many only happened when 2, 3, 4, etc. conditions were simultaneously true. We were surprised to find that no one had looked at this question empirically before. It turned out that all of the failures in the FDA reports appeared to involve four or fewer parameters. This was a very limited sample in one application domain, so I started looking at others and found a similar distribution. So far we have not seen a failure involving more than 6 parameters.

This does not mean there aren't any, but the evidence so far suggests that a small number of parameter values are involved in failures. In general, most problems are caused by one or two input values, fewer by an interaction among three, and still fewer at each step as we increase interactions above three.

What's interesting about this research is that it suggests we can significantly improve the efficiency and the effectiveness of software testing – if we can test all 4-way to 6-way interactions, we are likely to catch almost all errors, although there are a lot of caveats to that statement and we don't want to oversell this. In particular, selection of test values is critical unless a variable has a small set of discrete values, but this problem exists for all test methods. With the Automated Combinatorial Testing for Software (ACTS) project and tool, we hope to make these methods practical for real-world testing.

LogiGear: What has been your own personal contribution to this research and development? What sort of problems are you attracted to and how have you been able to solve them?

Mr. Kuhn: The series of papers that I did on the number of parameters involved in interaction failures has helped establish an empirical basis for the work and seems to have encouraged research on combinatorial testing by others.

I also developed a parallel algorithm for generating combinatorial tests, although, frankly, there hasn't been much need for it since Jeff Lei's IPOG algorithm in the ACTS tool is so fast. More recently I've worked out a way to apply combinatorial methods to testing event sequences – instead of n! Tests for n events, the number of tests is proportional to log n. This method was motivated by interoperability testing, and it has already been used successfully.

I've always been fascinated by large collections of data – whether there might be some interesting or useful properties hiding in a mass of numbers. This is one area where

that interest seems to have paid off. Another is some work I did showing that there's a hierarchy of fault classes for boolean expressions.

LogiGear: What do you think is the current state of this development? You're working on developing an open-source testing tool. How practical is it now? What successes have you experienced with it and what does it need? Also, how would you like its development to proceed in the future - would you like to see it take off and be developed as a sort of 'Linux OS' of combinatorial testing? What would you tell those who seek to get involved in this work?

Mr. Kuhn: The ACTS tool development is being led by Jeff Lei at U. of Texas Arlington, who has been an integral part of this project since the beginning. It's very robust and has been acquired by several hundred organizations, and so far the feedback is very positive. It appears to be easier to use and more efficient than other tools out there, but we want to integrate lessons learned and make it better for real-world projects. In particular, the constraint handling feature is being strengthened. This is what you need, for example, to prevent the generation of tests that specify Internet Explorer on a Linux system. It already has this ability, but we are learning new needs from users all the time.

I like your suggestion that ACTS could be like Linux, in that Linux has become both a freely available tool and the basis for successful commercial products and services. It would be great if ACTS followed a similar path. In fact we are already seeing commercial extensions of it.

LogiGear: And finally, where do you see your own contributions and questions heading in the future? Where do you see this group – this "movement" if you swill – headed?

Mr. Kuhn: Two things I really want to work on are integrating combinatorial testing with formal specifications and model based testing, and investigating the combinatorial coverage of tests developed with other methods. Both of these efforts should help to make this kind of testing more practical and cost effective. We have a good start on both, but we need better automation and tool support. The evidence so far suggests that we can make testing more efficient, but we need real-world data to prove it. I also want to understand the relationship between test value selection and combinatorial testing – are combinatorial methods more or less sensitive than other test methods to the choice of input values? But our primary goal is to collect and analyze data on real-world projects.

LogiGear: Thank you, Mr. Kuhn.

D. Richard Kuhn

Computer Scientist, National Institute of Standards & Technology

Expertise

- Combinatorial Testing
- Role Based Access Control
- Quantum Information Networks

Honors & Awards

- 2009: Excellence in Technology Transfer Award
- 2008: Best Standards Contribution, NIST/ITL
- 2007: Best Journal Paper Award, NIST/ITL
- 2002: Gold medal award for scientific/engineering achievement, U.S. Dept. of Commerce
- 1998: Excellence in Technology Transfer Award,1998
- 1990: Bronze Medal, U.S. Dept. of Commerce

"I've always been fascinated by large collections of data - whether there might be some interesting or useful properties hiding in a mass of numbers."

Dr. CEM KANER – Director, Center for Software Testing Education & Research, Florida Institute of Technology

Dr. Kaner shares his experiences on software testing and essential skills needed to be a software tester.

PC World Vietnam: What did you think of VISTACON 2010?

Dr. Kaner: I am very impressed that the event was very professionally organized and happy to meet my old colleagues to share and exchange more about our area of expertise. I was also very excited being one of the speakers at the conference and presented on some topics which focused on software testing as well as sharing some of my experiences from my research and work in this field.

PC World Vietnam: What do you believe are the most important skills required for a tester?

Dr. Kaner: Testers should have a good knowledge of the profession that they are working in; have testing skills and certain technology basics. For example, when testing banking or finance software, you should know about their trading activities, how they liquidate, and have a thorough grasp of some of the testing tools, testing plans and test automation methods. The technology basics means you should grasp some steps of the software development process, programming language skills, a knowledge of algorithms and operating systems. Besides that, other essential factors that can help you become a successful tester are having a good observant mind to easily recognize bugs during the testing process, the passion to apply the highest standards to the project, and finally, that you are always interested in new technology and new products.

PC World Vietnam: Could you please share with us your experience of undertaking testing projects in different countries?

Dr. Kaner: I have had the opportunity to work and collaborate on projects in the US and India in the field of software testing. From my experience, you should focus on the test cases created during the period you are creating code. Parallel to that, you need to improve the testing process to achieve the desired quality.

It is important to find ways to solve problems by preventing bugs instead of finding bugs. Testers should talk and share their experiences of finding bugs to improve their professional skills. Also, the wages paid to a tester in each country varies. For instance, in the US, the cost of hiring a tester is very high, so most companies outsource their projects to another country to save costs. According to Global Services Media, last year Ho Chi Minh City ranked 5th in the list top 50 emerging global cities in outsourcing software. That is a good sign for the development of the Vietnamese software industry.

PC World Vietnam: Vietnamese students often tend to become programmers instead of testers, what do you think about this?

Dr. Kaner: Actually, the hobbies and passions of each person will always be different. I myself have been rotated between the two positions of testing team manager and software programmers' team manager. Some people find the testing field interesting, but others discover their own skills are more in line with programming. Those who prefer to focus on solving a problem or specific solutions may be appropriate as a programmer. But if you decide you want to become a tester, you should do your best to make this happen by studying hard and doing serious research.

PC World Vietnam: Thank you, Dr. Kaner

(This interview is condensed and translated from the PC World Vietnam Magazine - October 2010 Issue)

Hard to Reproduce Bugs – Few Tips

By Santhosh S Tuppad

While testing, we sometimes find bugs and the next time we try to reproduce it again they are gone. But, our efforts to reproduce such bugs should depend on how critical the bug was. Investing time on very low severity bugs is sometimes not ideal depending on the context and your project demands. So you need to make wise decisions about whether you want to invest time on reproducing the bug or if you want to concentrate on other modules and find critical bugs. If you invest that much to reproduce a bug which is of low severity then you might miss the critical bugs which could easily be encountered by customers on release.

In this blog post I am going to tell you what you can do to replicate hard to reproduce bugs when you find them. However, I cannot assure you that with these points you will be able to reproduce the bugs.

Make a video recording while you are doing your testing activity – Watch the video to see what you did exactly and do the same steps. Things to remember while running video software:

1. Sometimes the performance issue might be due to the software you are running while testing the application. So, you might want to check the performance with and without the video software.

2. Choose good video software where you can watch the video frame by frame [Windows Media Encoder is one and there are plenty of others]

3. If you were not running video software, then you might want to use the power of your brain to go over exactly what you did. This is possible, and I have done this many times by forcing my mind to go back and reproduce the steps and finally reproduced the bug. Sometimes bugs make us feel that we found it through THESE steps but in between we did SOMETHING ELSE also which we forgot.

4. Keeping logs of what processes were running and what software was running when you encountered the bug. Having a timestamp when you found the bug and comparing it with the logs of what processes or software was running during that time would assist or help you in your investigation activity. There are various utilities to capture the logs – I won't give you the utility name but I recommend you explore them on your own which will help you to build your search capabilities.

5. Time Interval – Some bugs get reproduced after certain time intervals from this step to that step. Example: I clicked on "A" menu – and then "B" menu – I got error message. Then we try to follow the same thing but the bug is not reproduced. What exactly happened was when we clicked on "A" menu we were interrupted by someone and there was a 9 minute gap

and then we clicked on "B" menu which made that error message display. These are time-dependent bugs so you might want to record each and everything you did. Making notes is a skill and you might want to develop it to aid in your investigation activity.

6. Usage of Parental Software – This software can help you track each and every step you did on the application. It will record every action that you do. Example: You are testing "A" application and in between you do some actions on "B" application and you encounter an error message. So this will help you to record the steps such as after doing action on "A" application, what action you did on "B" application.

I do not hold any responsibility if you download any parental software and encounter some issues like data stealing etc. as some of the parental software includes viruses or Trojans that might steal your confidential data. I recommend you do research on particular parental software before you buy or download one.

Ask your peers to get involved in reproducing the bug – To brainstorm different ideas to reproduce the bug you can invite your peers to assist you as different testers have different ideas and they might be of help in replicating the bug. I hope this blog post of mine has helped you to some extent. Please do send me your comments if you have more ideas that were not covered. **Article from Tuppad's Blog**

Santhosh Tuppad biography

Over the last couple of years, Santhosh Tuppad has become known for his testing skills, winning bug battles and testing competitions across the world while being an avid blogger and testing enthusiast who organizes monthly meets for testers in Bangalore & Chennai.

Tuppad says he loves being hands on. While many of his peers were thinking about job security, Santhosh was game enough to start his own testing service, and he aims to provide an even greater service to the community in the future.

Santhosh got into testing because his girlfriend wanted to. He thanks his girlfriend so much today.

Santhosh blogs at http://tuppad.com/blog .

AUTOMATION TOOLS

What is Silk Test?

SilkTest® is a cost-effective, powerful tool for functional and regression testing – allowing you to create easy to maintain and fast executing test automation across a broad range of technologies to identify quality problems early on in the development lifecycle.

With SilkTest® you achieve consistent and repeatable results:

- Higher return on automation investments: Easy and efficient test automation for testers, developers and business analysts
- Completely automatic synchronization for testing asynchronous Web 2.0 applications with the ability to test across browsers with one script
- Shortened test cycles: The fastest execution of functional test scripts available so that the maximum amount of testing can be achieved in continuous testing windows
- Increased productivity: Advanced object recognition and synchronization for the highest reliability and lowest maintenance test automation for even the most demanding applications

Supported Technologies:

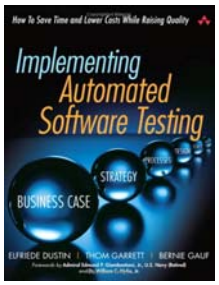
Adobe Flex, Java SWT, SAP, Windows API-based client/server (Win32), Windows Forms, Windows Presentation Foundation (WPF), xBrowser (Web applications)

Platform Support:

Windows XP SP3, Windows Vista SP1 or SP2, Windows 7, Windows 2008, Windows 2008 R2

SOFTWARE TESTING BOOKS

Implementing Automated Software Testing: How to save time and lower costs while raising quality



Authors: Elfriede Dustin, Thom Garrett, Bernie Gauf
Paperback: 368 pages
Publisher: Addison-Wesley Professional, 1 edition (14 March 2009)
Language: English
Product Dimensions: 9.2 x 7.3x 1.1 inches

Praise for Implementing Automated Software Testing

"This book fills a huge gap in our knowledge of software testing. It does an excellent job describing how test automation differs from other test activities, and clearly lays out what kind of skills and knowledge are needed to automate tests. The book is essential reading for students of testing and a bible for practitioners."
—Jeff Offutt, Professor of Software Engineering, George Mason University

"This new book naturally expands upon its predecessor, Automated Software Testing, and is the perfect reference for software practitioners applying automated software testing to their development efforts. Mandatory reading fo r software testing professionals!"
—Jeff Rashka, PMP, Coauthor of *Automated Software Testing and Quality Web Systems*

"Pragmatic and practical help for test automation"
By Lisa Crispin, Agile testing practitioner and coach.

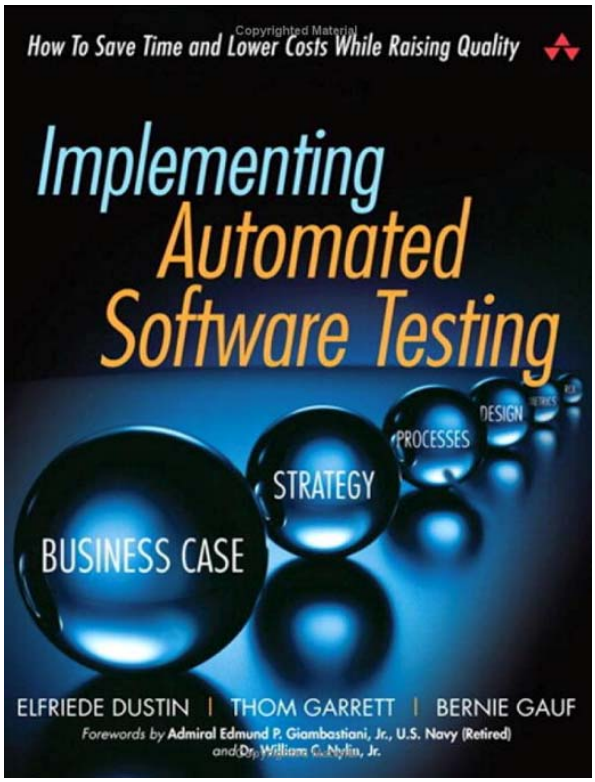
Amazon Reviews

It's a great resource, and will give readers a good grip on the fundamentals of test automation. I get so frustrated when people think it's impossible to automate, or that they have to hire some expensive consultant to get it done. This book will enable teams to be much more successful. It is a good overview of all the different areas where automation can help a team tremendously.

BOOK INTRODUCTION:

Implementing Automated Software Testing

Author: Elfriede Dustin, Thom Garrett, Bernie Gauf



Testing accounts for an increasingly large percentage of the time and cost of new software development. Using automated software testing (AST), developers and software testers can optimize the software testing lifecycle and free up time to do other tasks. As software technologies grow increasingly complex, AST becomes even more indispensable. While technologies continuously evolve, software testing techniques have stagnated somewhat and are the trailing edge of technology software implementations.

This book describes the importance of AST, building on some of the proven practices and the automated testing lifecycle methodology (ATLM) described in its predecessor book *Automated Software Testing (Addison Wesley, 1999)* and provides a renewed practical, start-to-finish guide to implementing AST successfully.

In *Implementing Automated Software Testing*(Addison Wesley, 2009, Dustin, Garrett, Gauf),three leading experts from IDT (http://www.idtus.com - a company solely founded for the purpose of "changing how testing is done" by developing edge of technology AST solutions and bringing software testing on par with edge of technology development), explain AST in detail, systematically reviewing its components, capabilities, and limitations. Drawing on their experience deploying AST in both the defense and commercial industries, they walk you through the entire implementation process—identifying best practices, crucial success factors, and key pitfalls along with solutions for avoiding them. You will learn how to:

- Make a realistic business case for AST, and use it to drive your initiative
- Clarify your testing requirements and develop an automation strategy that reflects them
- Build efficient test environments and choose the right automation tools and techniques for your environment
- Use proven metrics to continuously track your progress and adjust accordingly

Whether you're a test professional, QA specialist, project manager, or developer, this book can help you bring unprecedented efficiency to testing—and then use AST to improve your entire development lifecycle."

SOFTWARE TESTING CONFERENCES in December 2010

1	Asian Test Symposium	01 – 04	Shanghai, China	http://ats10.shu.edu.cn/
2	Workshop on RTL and High Level Testing	05-06	Shanghai, China	http://wrtlt10.shnu.edu.cn/
3	Test Expo	07	London, England	http://testexpo.co.uk/
4	International Conference on Software Engineering	06-07	Phuket, Thailand	http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=11048&copyownerid=13360
5	SIGIST - 'The Keynotes - 6 of the Best!'	08	London, England	http://www.bcs.org/server.php?show=nav.9264

MAGAZINE's CORNER

About Core Magazine

C0re Magazine is a free magazine for IT Experts in the field of Software Quality. It is a project of independent Testers and Quality Experts with the support of SJSI and gasq.
Our aim is to provide global platform for IT professionals to share knowledge and experience in quality area. We would like to present different points of view and different perspectives on quality.
Together with c0re Polish, there is also c0re English published. c0re English is dedicated for a global community.
In c0re you will find articles written by international experts, grouped within four basic sections:

- Software engineering - topics which cover requirements collecting and analysis, designing, software development life cycle, development methods, change management, configuration management etc.
- Software testing - practical aspects related to testing - techniques, methods, tools.
- Quality in project - quality assurance and control on the process level.
- Management - for those, who are involved in QA team leading, project and process management.
- Other sections: book's reviews, notifications about events and conferences, tools' reviews and comparisons, tips & tricks, feuilletons.

Our publisher, gasq (Global Association for Software Quality) is an international, member-based, non-profit association, a so-called "AISBL" (international association without financial interest).
Founding members of gasq come from all of Europe, but also from The Americas and Asia. The main goal of gasq is to support software quality in research, teachings and industry.
gasq maintains international networks and supports and develops certification programs, e.g. for software tester or usability professionals. At the same time, gasq is service provider for groups and associations that work for software quality. At the moment, gasq has offices in Belgium, Bulgaria, France, Germany and Spain. It is going to open offices in further countries soon.

SUBMIT YOUR ARTICLES TO LOGIGEAR MAGAZINE:

SUBMISSION GUIDELINES

LogiGear Magazine would like to announce that it is soliciting articles from writers, activists, journalists, and our readers who have an interest in writing about software testing topics. Material can include regular articles such as those in our magazine or from posted articles on blogs, websites or newsletters that you would like to reach a wider audience. These articles can be about: features, tips and hints, testing instructions, motivational articles and testing overviews that will assist our readers in gaining knowledge about software testing practices and the industry in general.

Please note:

1. Feature articles must be 500 words or more, Tips and Tricks must be 200 words or less, and Tool & Technology articles must be 500 words or more.

2. Only submissions from original authors will be accepted. By submitting this material, you have acknowledged that the material is legal and can be redistributed (book publishers or a public relations firm will need to reference this information at the top of the article). Such articles will not be compensated and will not be accepted if it can be interpreted as advertisements. However, you may place a brief resource box and contact information (but no ads) at the end of your article. Please specify if you wish to have your e-mail included for reader's response.

3. Due to staffing, editing of such submissions is limited; therefore, please send only the final draft of your work. After article is received, it may be revised before publishing, but not without your final approval.

4. Submissions may or may not be used. It is the sole discretion of LogiGear Magazine's editorial staff to publish any material. You will be notified in writing if your article will be published and what edition it will appear in.

Additional information about this process or to submit an article, please email: thuyen.vu@logigear.com