Test Methods and Strategies

Action-Based Testing, A New Perspective
By Michael Hackett

Black-Box Testing Techniques
Robin Roy

Why Exploratory Testing Should be Part of the Test Plan
Brian Heys

Key Principles of Test Design
Hans Buwalda
“Why do we need to understand a bunch of test methods? I write test cases from user stories or requirements, automate what I can and execute the rest manually, and its fine.” If this is your situation: good for you.

If you are time crunched, if your automated tests have lost relevance, are hard to maintain or regularly miss bugs, if you do not have useful and meaningful ways to confidently measure and report coverage and risk, if you are doing what you have always done, if you document too much, or document too little, if testing is “a mess”, if the dev teams do not trust your testing, if you wish there was a better way......then arming yourself with new test methods or examining the methods you currently use will, without a doubt, be beneficial!

In our continuing effort to be the best source of information for keeping testers and test teams current, we have dedicated this issue to exploring test methods. Learning test methods is core to a test engineer. These are the skills and methods we use in the daily execution of our work. Without having enough tools in our arsenal, our job is compromised. Worse, it can be inefficient, insufficient, misleading and worst- miss bugs!

From useful and complex Linear Code Sequence and Jump (LCSAJ), to old faithful, Model-based testing, there are a very large number of important test methods. Lately I have seen a renewed use and importance of Scenario-based testing using personae for higher customer satisfaction, Real-world testing and user story validation.

Why does learning more test methods help? Test methods provide a structure for thinking. They give a framework for well understood measures of coverage and risk. For example, Model-based testing can give easy measurements of path coverage, Requirements-based testing is a common method when requirements coverage is measured or there is a need for regulatory compliance.

Different methods and techniques have different uses and goals. Different goals might be, for example, finding bugs, customer satisfaction, regulatory compliance, getting the product out as fast as you can, having confidence in the already functioning parts of the product with new added functions- all need different methods to provide the greatest confidence for these different goals.

Using particular test methods in your test strategy takes away the seemingly random nature of some test teams. I know a few teams who “hope” they find the worst bugs. Remember, hope is not a strategy! Test methods help create your strategy!

The objective of this edition is to present some new views on test methods. The goal is to give you as many tools as possible to attack your test effort and do the most effective, efficient job and communicate it effectively to the team so you can make the best most informed decisions on bug fixing, risk and release!

In this issue Brian Heys warns that without exploratory tests, the number of defects will always be higher; I’ll explain how Action-Basted Testing is a much saner way to evolve a testing project; Salesforce’s Keith Stobie reviews, “The Domain Testing Workbook” by Cem Kaner, Sowmya Padmanabhan and Douglass Hoffman; Robin Roy writes that boundary guidelines can provide a higher rate of error detection and LogiGear CTO, Hans Buwalda explains that a good test design can improve quality and the efficiency of the tests.

And, at this time of year, for those of you who celebrate Lunar New Year- Happy Year of the Horse! For those of you who do not celebrate Lunar New Year- give it a try! Its as good a reason as any to have a fun celebration.

Michael Hackett
Senior Vice President, LogiGear Corporation
Editor in Chief
4 IN THE NEWS

5 WHY EXPLORATORY TESTING SHOULD BE PART OF THE TEST PLAN

Brian Heys

Without exploratory tests, the number of defects that find their way into production will always be higher.

7 ACTION-BASED TESTING, A NEW PERSPECTIVE

Michael Hackett, LogiGear Corporation

The best part of ABT is that it is for thinking people. It is intelligent and creative. It is a much saner way to evolve a testing project.

11 BOOK REVIEW

Keith Stobie, Salesforce


13 BLACK BOX TESTING TECHNIQUES

Robin Roy

By applying these guidelines, boundary testing will be more complete, thereby having a higher likelihood for error detection.

16 KEY PRINCIPLES OF TEST DESIGN

Hans Buwalda, LogiGear Corporation

Regardless of the method you choose, simply spending some time thinking about good test design before writing the first test case will have a very high payback down the line, both in the quality and the efficiency of the tests.

18 TEST METHODS AND STRATEGIES GLOSSARY

Some of the terms used when discussing test methods and strategies.

19 VIETNAM VIEW: THE LARGEST CAVE IN THE WORLD

Brian Letwin, LogiGear Corporation

With the discovery of the world’s largest cave early last year, international visitors have been putting down huge sums of cash to explore its forests and waterfalls.
In the News

Video: The Importance of Regression Testing

JoEllen West says that development shops must conduct thorough Regression testing to mitigate the instances where a change to the codebase breaks a feature in the software. Iterative development, Unit testing, Automated testing, Continuous Integration, and testing early and often are the more accepted design practices in regression testing. Regression testing can be either continuous or you can have a regression testing iteration at the end of every development cycle.

West says it’s also preferable to automate many regression tests, especially if you’re using the Continuous method. West concludes by explaining how a manager will know if their regression testing strategy is working. You can watch the interview here: http://agile.dzone.com/videos/importance-regression-testing.

Despite Improved Patch Times in 2013 Vulnerability Battle Goes On

Software vendors have improved their response to security flaws in the last 12 months but some still take too long to patch the highest-risk vulnerabilities, figures from Swiss testing firm High-Tech Bridge have suggested.

Comparing 2012 to last year, the firm found that critical flaws were now being patched in 11 days (up from 17), while medium and low-risk flaws were now being fixed in 13 and 25 days respectively (as against 29 days and 48 days).

This means that the average time to patch has fallen across categories from 27 days to 18 days, a 33% improvement.

These statistics are based on the 62 security advisories released by High-Tech Bridge through its ImmuniWeb SaaS testing service, covering 162 vulnerabilities, so the reported improvement is indicative rather than definitive.

You can view the full report here: http://www.techcentral.ie/despite-improved-patch-times-in-2013-vulnerability-battle-goes-on/

LogiGear Celebrates 20 Years of Testing Excellence and Development

This year marks LogiGear’s 20 year anniversary. Hung Nguyen, Michael Hackett and Heather Ho cofounded the company in 1994 to provide software testing consulting and training programs to up and coming Silicon Valley companies.

Today, LogiGear has close to 1,000 employees and a portfolio that includes software development and test automation services.

“We’re pleased to have achieved this milestone, and sincerely thank our loyal partners and customers who have trusted in us,” says Hung Nguyen, LogiGear CEO. “We’re all looking forward to the next 20 years as we continue helping companies accelerate their growth by giving them confidence in the software they deliver.”
Why Exploratory Testing Should be Part of the Test Plan

A test plan should always include exploratory tests as part of the approach. Without them, the number of defects that find their way into production will always be higher.

By Brian Heys

Encourage Learning

According to the Wikipedia entry: *Exploratory testing ... is concisely described as simultaneous learning, test design and test execution.*

The key word is learning. One of the problems with scripted tests and script reuse is that the same functionality is covered, over and over again. This inhibits learning by narrowing the tester’s focus only to known areas, discouraging them from going off on their own and exploring other features or other ways of using the software.

People like to learn, and exploratory testing encourages that, making the process of testing more challenging and enjoyable, and leading to more productive testing.

“The real problem is not that the software hasn’t been tested, but that only key parts have been tested.”

Uncover Defects Earlier

A big problem with only running scripted tests is that they cover the same ground. While this may be fine for regression testing of core func-
tionality, such script reuse does not fully test software changes in the way thorough exploratory testing can.

Relying only on scripted tests results in a larger than desired number of defects being missed during UAT.

As a tester, I have lost track of the number of times I have heard business users complain during UAT that “nobody has tested this”, when the real problem is not that the software hasn’t been tested, but that only key parts have been tested.

End-users are often very effective at finding bugs – in part this is probably due to their tendency to take a more exploratory approach as a result of their non-familiarity with the application.

Suggested Metrics for Exploratory Testing

Exploratory testing is sometimes left out of test plans because of uncertainty over what testing is being carried out, and therefore how to report progress. Many test managers focus on numerical progress reporting because of expectations from project and program managers who like to see a dashboard-style report that clearly shows the status of testing.

When there is no set number of planned exploratory tests to be carried out it can be difficult to show numerically whether or not you are on track, or whether you are achieving sufficient coverage of functionality.

One solution is to allocate a number of days for exploratory testing, and report a percentage complete based on the time elapsed. The level of coverage could be reported alongside that by completing a coverage matrix as tests are executed in defined areas of functionality.

Summary

However progress is reported, the key takeaway here is that a test plan should always include exploratory tests as part of the approach. Without them, the number of defects that find their way into UAT and production will always be higher.

By scheduling a number of days into the test plan for exploratory tests, more defects can be uncovered earlier in testing, sometimes dramatically improving the perceived quality of what gets handed over to the users.

About Brian

Brian is a delivery-focused manager with a wealth of multi-disciplinary IT experience, the last fifteen years of which have been mainly in quality assurance and testing. He has been fortunate enough to work for some of the world’s largest companies using some of the latest technologies and tools.

As well as being responsible for the day-to-day running of the business, Brian keeps a hand in the industry by being regularly involved in projects as an interim manager. He also runs The Naked Tester, a regularly updated blog, sharing curated and created content about testing and quality assurance and all other fun things in life.
Action-Based Testing, A New Perspective

The best part of Action-Basted Testing is that it is for thinking people. It is intelligent and creative. It is a much saner way to evolve a testing project.

By Michael Hackett, LogiGear

All testers and quality engineers hear about Action-based testing (ABT) or keyword-driven testing somewhere. There are automation tools focused on keywords and actions. Maybe people have read an article about it. But do you actually do ABT as a test method? Probably not. What’s the big deal with ABT? Why don’t more teams do it?

ABT sounds good to use as a method. I often wonder: “what do people really know about Action-based testing?” I hear: “Isn’t ABT about making a bunch of small, modular automated test cases—and calling them actions, then reuse and string them together into bigger scenarios? Isn’t that ABT?” Actually not. That would be using action words, or just keywords for that matter for your automation without using ABT as the test method to design and develop your tests.

Here are some more examples of missing the point of ABT. To me, some of the most important parts of Action-based testing are:

- It all starts with test design. Stop starting a test project by writing mindless test steps and more steps and more steps- the maintenance cost is too high.
- ABT is iterations of greater specificity—start high and build into lower levels of detail.
• ABT is separation of test design and test execution. Even though it may wind up being done by one person, the test designer/subject matter expert hat is worn first. Automaton can be specified later, by the same person or someone with different skills.

• Focus on intelligent test design and not mindless test execution. It is more important to focus on what is going to get tested rather than how those tests will get executed.

• ABT easily enables greater sharing of test cases, faster test documentation, better division of work, and helps other team members by not repeating the same actions over and over. Its more human-easier sharing, easier and better communication.

So why don’t more teams use the ABT method? I think it is difficult for many teams to break away from the “what we have always done” mindset. It’s difficult to get team or management buy-in on testing not focused uniquely on requirements. Plus, too many people think ABT is only about automation.

This article focuses on the infrequently discussed aspects of ABT— the thinking and human side of the method. Action-based, or keyword-based testing has become so popular with automation tools that many people think it is just a method of automation. It’s true that ABT will give you more, better and easier test automation and you can find that written about in many places. I want to focus on the areas of ABT I like that are independent of test automation.

ABT in a Nutshell

ABT is much more than creating action words! It’s a method to organize and design testing by focusing on test modules and test objectives rather than how the tests will be executed. ABT emphasizes thoughtful design and focuses on what you are testing rather than on how to script tests. Trying to accomplish too much with detailed scripting is what usually results in high maintenance. ABT creates easier to understand documentation, better test coverage, and easier, more scalable test automation with the full participation of both testers and automation (coding) engineers.

Too often ABT is reduced to merely making a bunch of reusable actions and modular testing. That is not ABT, the actions are definitely a byproduct and huge benefit of ABT, but if it stops there, you are missing the point and biggest benefits of ABT.

Test Design—Think, Think, Think

Some automation tools lend themselves to mindless scripting that produces high maintenance, low re-use, non sharable test scripts. Bad automation. ABT, to me, is about test design. Test Design is the process of analyzing test requirements and arriving at the test modules, test objectives and then test cases. It starts with an analysis of what areas will be tested and the creation of modules. A test module is a way to organize test cases into logical, related functions. A test module is a much larger unit than a test case, and contains multiple test cases.

The test design process is thinking about what you are testing and examining the product you are about to develop in order to define what has to be tested.

In ABT, it is crucial to have a good high level test design in order to effectively define test modules. The modules and test objectives are where we see the test design, not in the test steps or cases. Each test module should have a clear scope that is different from the others to avoid redundancy. Each module should be self-contained and to avoid interdependency to reduce maintenance and provide for easier distribution of work.

A test module consists of test objectives and action lines, and will resemble a spreadsheet. The test objectives outline the scope of the test module into individual verbal statements defining what needs to be tested in the module. Test objectives are my favorite part of ABT. They are the ideas, risks, objectives—things I want to test. Anyone can easily read these sentences and get the complete understanding of what I am testing rather than wading through endless steps and steps and steps.

The tests within the modules are defined by a series of action lines, often further organized in one
or more test cases. Every action line consists of an action word that defines the test step, and arguments that define the data for the action. Arguments can include input values and expected results.

Not only will good test design result in good coverage, it is also a major contributor to efficiency. The principle of test design should be "lean and mean." The tests should be of a manageable size, and at the same time complete and aggressive enough to find bugs before a system or system update is released.

Some people define test design as specifying test cases. I disagree. Test design is the process of thinking, using your brain, organizing the project, identifying high level ideas on what has to be tested, without the specifics. A test case is too small and too isolated of a unit to give good direction to test development. Rather than having a predefined list of test cases to be developed, I like to make a list of test modules, and let the test cases in them be the result of test design, not the input of it.

After the test modules are defined and taking a first pass on the test objectives, you can begin assembling test cases and actions. You create test cases to execute the objectives, but create them by using action words as opposed to long sets of steps that are hard to read and very soon become cumbersome and then require a great time investment to create and maintain.

**Focus on What, Not How**

From management practices to self-improvement to software programming best practices, *Focus on what, not how* is the effective mantra, and its the core of ABT.

Focusing on what you test is the beauty of ABT, how you test it comes later. What you are testing dictates test design—it’s creative, imaginative, thoughtful and intelligent. How you will test is a different issue—it’s engineering, or simple, step-by-step robotics. Yes, the how may require creative intelligent solutions, but with ABT, this is less common.

Don’t focus too much on details in the beginning. Start with big picture, how you test and lower level details get resolved in later iterations of planning. Immediately scripting step by step is not a sophisticated approach. Though it is common, its a bad idea!

What is now. How comes later. *What* — the goals and success factors and coverage; *How* — low level, detailed later in the process. Letting go of the how until later also unblocks your mind and allows ideas and solutions to flow to you.

**ABT on Agile Teams**

Starting with what has to be tested and building increasing levels of detail fits perfectly into the agile mindset. For estimating the time for testing in a sprint planning session, it is more important to focus on what has to be tested than how it will be tested. Are there cases where how something is tested greatly impacts the test estimate? Yes, but rarely. Too often the extent of testing, the variety of testing and the ranking of tests is ignored in favor of how long an individual test takes to run in giving estimations.

During release planning, at the beginning of a set of sprints and at sprint planning meetings, the features, UI, and implementation is not yet planned. How can you plan the execution of the test? You can’t without a high maintenance cost. What you absolutely can do is plan out and design what has to be tested the areas, modules and ideas, objectives to be tested. You know that features have to be tested in a specific way, on already defined
platforms, with known data, using certain interfaces—its test design. This can be a big asset in conveying risk, coverage, estimates, your thinking and needs for the testing project in language the scrum team can understand—without focusing on how the test will be executed.

The most common practice is, and will remain, to develop and execute tests as part of sprints. In a sprint, functionality is progressively understood from user stories and conversations to become clear enough for testers to test it. This can be done in developed tests similar to ABT test modules, as well as exploratory and interactive testing. It can also be good practice to capture at least some of the “interesting” interactive tests in test modules for future use.

Have as many business level tests as possible, as they add great value to overall depth and quality, as well as being resilient against system changes that do not pertain to them. Use the high level test design step in ABT (where the test modules are identified) to determine what can be done early on in business level tests, and what needs to be completed in detail tests as part of development sprints.

Benefits of ABT

ABT is more human. It’s easier to communicate. Easier to share actions and test cases. It enables better test understanding and team collaboration. When the automation finally gets specified as actions, the individual actions can easily be shared and reused, making writing test cases easier, faster and, lower maintenance.

ABT Improves the organization of tests. Instead of randomly grouping tests together, each module has a clear scope that tests must fall under which gives better visibility into testing. It also enhances business functionality test coverage. For example, each module can cover a major functional requirement (e.g. “Make Reservation” or “Cancel Reservation”) or a non-functional requirement (e.g. Scalability Test that registers 1 million users)

Organizationally, the big advantage with ABT is scalability. Unlike with coding, using actions eliminate redundant steps and automation. This makes it faster to create new tests, so there is less than 1:1 effort required for testing as the complexity of the software increase, so you can quickly create a large number of tests.

Is ABT perfect? No. Is it better than what the vast majority of teams do? Absolutely. There is a ramp-up curve to learning the details and implementation nuances of ABT. Defining modules, high-level and low-level action, are skills to be built. While starting a test project with step-by-step test cases is certainly faster, it has less intelligence, higher maintenance, more work, with low visibility.

Also—you use your brain for ABT. You get to think, envision what needs to be tested early in the project. I see this as very positive as opposed to the boredom of creating test steps.

In summary, to me, the big payoff with ABT is the speed at which I can document a test project. I can’t stand the step-by-step, brainless mess. I do not like working on mindless projects.

Yes, there is a huge payoff when it comes to automation—clearly. But the best part of ABT is that it is for thinking people. It is intelligent and creative. It is a much saner way to evolve a testing project.

About Michael

Michael Hackett co-founded LogiGear in 1994 and leads the company’s LogiGear University training operations division, setting the standard in software testing education programs for many of the world’s leading software development organizations.

Michael is co-author of the popular Testing Applications on the Web (Wiley, 2nd ed. 2003), and has helped many clients produce, test and deploy applications ranging from business productivity to educational multimedia across multiple platforms and multiple language editions.
**Book Review**

Book Review of The Domain Testing Workbook

By Keith Stobie, Salesforce

The Testing Domain Workbook is the most extensive and exhaustive work you will ever find on a specific testing technique (or related techniques if you include equivalence class analysis and boundary testing as the book does).

What I like best is the combination of academic background and roots combined with practical experience and industrial practice. All the concepts are presented in a simple and approachable manner with pointers to more details for those desiring more.

While the book appears daunting in size, it is only because of the extensive examples and exercises. The core of the book is very approachable and less than 100 pages. To gain mastery, working through the exercises is most useful, but you can do that over time.

Many practical aspects and considerations for testing are covered that are usually skipped over in broad testing surveys or short articles. For example, many books talk about different approaches such as risk-based, or pair-wise testing. Books may also cover the issue of combining values for a test, but Testing Domain Workbook walks you through the details and implications of what each approach entails when applied to combining values for a domain test. Further, it provides extensive guidance of when (in which context) the advice is most applicable (or not) such as:

*If you’re doing system testing after the programmers have done extensive unit testing of their variables, it will be unnecessary and wasteful to do thorough testing of secondary dimensions.*

The book incorporates many viewpoints, sometimes strong opinions, and pithy statements such as:

*Boundaries are funny things. When people say “No one would need a value that big,” what they really mean is “I can’t imagine why anyone would need a value that big.” The world is often less constrained than the limits of our imagination.*

The book is exacting and consistent in its terminology, but the reader needs to be care-
ful to keep the concepts clear and distinct. For example:

Well-designed domain tests are powerful and efficient but aren’t necessarily representative. Boundary values are suitable for domain testing even if those values would be rare in use.

The best representative of the class is the one that makes the most powerful test.

So the best representative, most powerful, is not necessarily the most representative of typical values. The book focuses on boundary values and bug hunting so that typical values are unlikely to be used even though they are part of the domain. You need to use more than the one well-developed technique of this book as the authors themselves state. For example:

well-designed scenario tests are usually representative but they’re often not powerful. To test a program well, you’ll use several different techniques.

You may become a better tester if you read this book. You will become a much better test-
er if you actually work through the exercises of the book. ■

About Keith

Keith Stobie is a Quality Engineer Architect at salesforce.com who specializes in web services, distributed systems, and general testing especially design. Previously he has been at Tivo and for Bing Infrastructure where he planned, designed, and reviewed software architecture and tests. In Microsoft’s Protocol Engineering Team he worked on Protocol Quality Assurance Process including model-based testing (MBT) to develop test framework, harnessing, and model patterns. With three decades of distributed systems testing experience Keith’s interests are in testing methodology, tools technology, and quality process. Check out his blog (http://testmuse.wordpress.com) to learn more about his work.

Perceived Importance of Software Quality Assurance by CIO

Source: Original Software

- 44% CIO’s indicating software quality IS Very important
- 25% Nice to have
- 16% CIO’s indicating software quality is a cost of doing business
- 15% Not defined
- 40% CIO’s indicating software quality ISN’T that important
- 29% Fundamental business process
- 15% Strategic business imperative
Software is tested from two different perspectives:

1. Internal program logic is exercised using “white box” test case design techniques.
2. Software requirements are exercised using “black box” test case design techniques.

In both cases, the intent is to find the maximum number of errors with the minimum amount of effort and time.

Black-box testing alludes to tests that are conducted at the software interface. Although they are designed to uncover errors, black-box tests are used to demonstrate that software functions are operational, that input is properly accepted and output is correctly produced, and that the integrity of external information (e.g., a database) is maintained. A black-box test examines some fundamental aspects of a system with little regard for the internal logical structure of the software.

Black Box Testing Techniques

Black-box testing, also called behavioral testing, focuses on the functional requirements of the software. That is, black-box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. Black-box testing is not an alternative to white-box techniques. Rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods.

Black-box testing attempts to find errors in the following categories:

1. Incorrect or missing functions.
2. Interface errors.
3. Errors in data structures or external database access.
4. Behavior or performance errors, and
5. Initialization and termination errors.

Unlike white-box testing, which is performed early in the testing process, black box testing tends to be applied during later stages of testing.

Because black-box testing purposely disregards control structure, attention is focused on the information domain. Tests are designed to answer the following questions:

By Robin Roy

Most software engineers intuitively perform BVA to some degree. By applying these guidelines, boundary testing will be more complete, thereby having a higher likelihood for error detection.
• How is functional validity tested?
• How is system behavior and performance tested?
• What classes of input will make good test cases?
• Is the system particularly sensitive to certain input values?
• How are the boundaries of a data class isolated?
• What data rates and data volume can the system tolerate?
• What effect will specific combinations of data have on system operation?
• There are various techniques in developing test cases for black box testing. I will be explaining two of the most common techniques, Equivalence Partitioning and BVA (Boundary Value Analysis).

**Equivalence Partitioning**

Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived. Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition. An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is either a specific numeric value, a range of values, a set of related values, or a boolean condition. Equivalence classes may be defined according to the following guidelines:

1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
3. If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
4. If an input condition is boolean, one valid and one invalid class are defined.

As an example, consider data maintained as part of an automated banking application.

The user can access the bank using a personal computer, provide a six-digit password, and follow a series of typed commands that trigger various banking functions. During the log-on sequence, the software supplied for the banking application accepts data in the form:

- Area code—blank or three-digit number.
- Prefix—three-digit number not beginning with 0 or 1.
- Suffix—four-digit number.
- Password—six digit alphanumeric string.
- Commands—check, deposit, bill pay, and the like.

The input conditions associated with each data element for the banking application can be specified as area code:

- Input condition, Boolean—the area code may or may not be present.
- Input condition, range—values defined between 200 and 999, with specific exceptions.
- Prefix: Input condition, range—specified value >200.
Input condition, value—four-digit length.

Password: Input condition, Boolean—a password may or may not be present.

Input condition, value—six-character string.

Command: Input condition, set—containing commands noted previously.

Applying the guidelines for the derivation of equivalence classes, test cases for each input domain data item can be developed and executed. Test cases are selected so that the largest number of attributes of an equivalence class are exercised at once.

**Boundary Value Analysis**

BVA extends equivalence partitioning by focusing on data at the “edges” of an equivalence class.

For reasons that are not completely clear, a greater number of errors tends to occur at the boundaries of the input domain rather than in the “center”. It is for this reason that boundary value analysis (BVA) has been developed as a testing technique. Boundary value analysis leads to a selection of test cases that exercise bounding values.

Boundary value analysis is a test case design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the "edges" of the class. Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.

Guidelines for BVA are similar in many respects to those provided for equivalence partitioning:

- If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b and values just above and just below a and b.
- If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.
- Apply guidelines 1 and 2 to output conditions. For example, assume that a temperature vs. pressure table is required as output from an engineering analysis program. Test cases should be designed to create an output report that produces the maximum (and minimum) allowable number of table entries.
- If internal program data structures have prescribed boundaries (e.g., an array has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.

Most software engineers intuitively perform BVA to some degree. By applying these guidelines, boundary testing will be more complete, thereby having a higher likelihood for error detection.

**About Robin**

Robin works as a Senior Consultant with Brilliance MSC, Malaysia.

He loves to evaluate new technologies and implement the same.

Believes in sharing knowledge.
**Key Principles of Test Design**

Regardless of the method you choose, simply spending some time thinking about good test design before writing the first test case will have a very high payback down the line, both in the quality and the efficiency of the tests.

By Hans Buwalda, LogiGear

---

Test design is the single biggest contributor to success in software testing and it is also a major factor for success in test automation. This is not that intuitive. Like many others, I initially thought that successful automation is an issue of good programming or even "buying the right tool". That test design turns out to be a main driver for automation success is something that I had to learn over the years, often the hard way.

What I have found is that there are three main goals that need to be achieved in test design. I like to characterize them as the "Three Holy Grails of Test Design", a metaphor based on the stories of King Arthur and the Round Table. Each of the three goals is hard to reach, just like it was hard for the knights of King Arthur to find the Holy Grail. This article will introduce the three "grails" to look for in test design.

The terminology in this article and is based on Action-Based Testing (ABT), LogiGear's method for testing and test automation. You can read more about the ABT methodology on the LogiGear web site.

**The Three Goals for Test Design**

The three most important goals for test design are:

- **Effective breakdown of the tests**
  
  The first step is to breakdown the tests into manageable pieces, which in ABT we call "test modules". At this point in the process we are not yet describing test cases; we simply identify the "chapters" into which test cases will fall. A breakdown is good if each of the resulting test modules has a clearly defined and well-focused scope, which is differentiated from the other modules. The scope of a test module subsequently determines what its test cases should look like.

- **Right approach per test module**
  
  Once the breakdown is done each individual test module becomes a mini-project. Based on the scope of a test module we need to determine what approach to take to develop the test module. By approach I mean the choice of testing techniques used to build the test cases (like boundary analysis, decision tables, etc.), and who should get involved to create and/or assess the tests. For example, a test module aimed at testing the premium calculation of insurance policies might need the involvement of an actuarial department.

- **Right level of test specification**
  
  This third goal is where you can win or lose most of the maintainability of automated tests. When creating a test case try to specify those, and only those, high-level details that are relevant for the test. For example, from the end-user perspective "login" or "change customer phone number" is one action; it is not necessary to specify any low-level details such as clicks and inputs. These low-level details should be "hidden" at this time in separate, reusable automation functions com-
mon to all tests. This makes a test more concise and readable, but most of all it helps maintain the test since low-level details left out will not have to be changed one-by-one in every single test if the underlying system undergoes changes. The low-level details can then be re-specified (or have their automation revised) only once and reused many times in all tests. In ABT this third principle is visible in the "level" of the actions to be used in a test module. For example, in an insurance company database, we would write tests using only "high-level" actions like "create policy" and "check premium", while in a test of a dialog you could use a "low level" action like "click" to see if you can click the OK button.

Conclusion

Regardless of the method you choose, simply spending some time thinking about good test design before writing the first test case will have a very high payback down the line, both in the quality and the efficiency of the tests.

About Hans

Hans Buwalda leads LogiGear’s research and development of test automation solutions, and oversees the delivery of advanced test automation consulting and engineering services. The original architect of the keyword framework for software testing organizations, he assists clients in strategic implementation of Action Based Testing™ throughout their testing organizations, and he is lead developer of LogiGear’s TestArchitect™, the keyword-based toolset for software test design, automation and management.

Notable Quotes

“Following scripted instruction is a great way of leaving you within the confines of the script.”

Cem Kaner
Software engineering professor, Florida Institute of Technology

“Sometimes familiarity with the technical details of a system can hide problems that are obvious to those that don't know the technology, the requirements documents, and the test scripts. As testers it is important that we be careful not to let our familiarity with a system make us blind to bugs -- things that bug our users.”

Ben Semo
Tester, skeptic, and defensive pessimist.
Test Methods and Strategies Glossary

**Test Strategy**
A test strategy describes how the test effort will reach the quality goals set out by the development team.

Sometimes called the test approach, test strategy includes, among other things, the testing objective, methods and techniques of testing and the testing environment.

Test strategies describe how the product risks of the stakeholders are mitigated at the test-level and which types of test are to be performed. The strategy can be documented on its own or is more commonly included as part of a test plan. Test strategies layout plans for how much manual and automated testing will happen at what phases and using which methods, such as unit testing, code review, UI validation, API testing, regression testing, etc.

**A Test Plan is...**
A document describing the scope, approach or strategy, resources, and schedule of intended testing activities. It defines test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.


**Test Design is...**
Thinking about what you are testing and examining the product you are about to develop. It is the process of analyzing test requirements and arriving at test objectives and then test cases. It starts with an analysis of what areas will be tested then later, how they will be tested.

Test design could require all or one of:

- Knowledge of the software, and the business area it operates on.
- Knowledge of the functionality being tested.
- Knowledge of testing techniques and heuristics.

**Test Methods**
Test methods are structures, ideas, or approaches to how you will design the test cases to execute the test strategy. Test methods may be determined by standards, regulatory agencies, or contractual agreements. They may be dependent on available documentation or lack of documentation. Test methods often go hand-in-hand with the test strategy as how you will design and execute the test cases.

*Black Box Testing* — The technique of testing without having any knowledge of the interior workings of the application. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system’s user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

*White Box Testing* — The detailed investigation of internal logic and structure of the code. White box testing is also called glass testing or clear box testing. In order to perform white box testing on an application, the programmer or tester needs to possess detailed knowledge of the internal working of the code.

*Grey Box Testing* — A technique to test the application with limited knowledge of the internal workings of an application, system, or platform. In software testing, the term *the more you know the better* carries a lot of weight when testing an application.

**Coverage**
Test coverage is a measurement of the extent of testing based on some criteria. Common measurements are code coverage, platform coverage, requirements coverage, user story coverage, form coverage, data coverage. There are many ways to measure coverage and report back to the team on testing progress and confidence.

Sources: Wikipedia, Tutorials Point
Vietnam View

Son Doong: The World’s Largest Cave

With the discovery of the world’s largest cave early last year, international visitors have been putting down huge sums of cash to explore its forests and waterfalls.

By Brian Letwin, LogiGear Corporation

Vietnam’s beaches and mountains are popular draws for tourists visiting Vietnam but there’s one attraction that takes the cake – Son Doong (which literally translates to Mountain River), the world’s largest known cave.

The Son Doong cave was discovered in 1991, but the steep drop into its mouth scared locals away. According to lore, locals were afraid of the cave because of the shrill whistling sound made by its fast-flowing underground river.

British cavers were the first to explore the cave in 2009. And what they found was like nothing they’d ever seen — not only does the 5 mile-long cave sport numerous waterfalls and even a forest, but at one point it’s tall enough to fit a 40 story skyscraper.

As one can imagine, the forest within the mouth of the cave supports a web of life and the cave itself contains many unique geological features. Monkeys enter the cave to eat snails and poisonous centipedes are found in abundance throughout. Abnormally large cave beads the size of baseballs have been found on its floor, and huge drops and unusually sharp rock formations are plentiful.

Small tour operators now have government approval to guide travelers through the Son Doong cave. Last August, the first group of individuals from Australia, Norway, Russia, the United States and the UK paid $3000 each to trek through the cave.

The beauty and diversity within the cave adds to the appeal as well as the risks of exploring. In spite of the risks, the remoteness and high cost, there’s a long waiting list explore the cave. Or maybe these are the things that explain its newfound popularity.

Vietnam is truly full of surprises.