DEDICATED TO SHOWCASING NEW TECHNOLOGY AND WORLD LEADERS IN SOFTWARE TESTING

LogiGear MAGAZINE

THE TEST METHODS & STRATEGIES ISSUE

Mobile Application Testing: Process, Tools & Technique

By Lindiwe Vinson

2011 GLOBAL TESTING SURVEY

Test Methods, Tools & Metric

By Michael Hackett

RELATED ARTICLE

User Stories, Scenarios & Use Cases

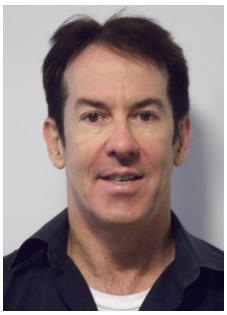
By Nadine Schaeffer

SPOTLIGHT INTERVIEW

Janet Gregory, Co-author of *Agile Testing: A Practical Guide for Testers & Agile Teams*

July 2011 | Volume V | Issue 5

EDITOR'S LETTER



How do you test software? How do you validate it? How do you find bugs? These are all good questions anyone on your project team or anyone responsible for customers may ask you. Can you articulate your test strategy—not your *test process*, but explain your approach to testing? I find that this can be a great challenge for test teams.

This month's theme naturally follows the Test Process Improvement Issue as it is common to identify a gap in testing skills, test methods or strategies during process improvement. This issue provides you a refreshing look at various traditional test methods.

Many test teams test out of old habits. However, today's fast-moving and high-stakes software development world demands teams *do more with less*, reduce risks and *be more agile*—and do all this faster! These pressures require the ability to effectively communicate your strategy and risks.

This month's issue addresses something helpful to me: I can fall into a rut and need to re-examine how I test. I'd like to find out how other people are practicing. Hearing how others handle testing challenges improves my own testing. Executing the same tactics will not address new issues—I need new ideas!

Our issue provides interesting vantage points and new concepts with progressive authors discussing current topics. Lindiwe Vinson reports on Organic Inc's strategy in mobile testing, an area of tremendous growth in development and testing today. Nadine Shaeffer evaluates the basics of writing good user scenarios and distinguishing between user story, user scenario, and use case. Janet Gregory, co-author of *Testing in Agile* answers test practitioners' questions on test methods and strategy for Spotlight Interview; and Blogger of the Month features Jason Barile pointing out key features in a test plan. Also included in this issue is also the latest installment of the 2010 Global Test Survey, presenting results in the areas of test methods, tools and metrics.

With a mixture of both new and traditional testing tactics, we've selected a line-up of articles that will address testing methods and strategies of today.

Our next issue will focus on outsourced and/or offshored testing projects. Stay tuned!

Michael Hackett
Senior Vice President
Editor in Chief

IN THIS ISSUE

BLOGGER OF THE MONTH

Jason Barile

Blog: Testing Testing

I've been reviewing a *lot* of test plans recently. As I review them, I've compiled this list of things I look for in a well written test plan document. Here's a brain dump of things I check for, in no particular order, of course, and it is by no means a complete list.

That said, if you see a major omission, please chime in to let me know what you expect from or include in your test plans. These requirements may or may not work for you based on your development methodology, but it's what I look for. I'm sure I'll think of several more points as soon as I publish this post!

Test Strategy

I always like to see the testing strategy called out early in the test plan. In general, it's always best to vet your testing strategy with your team as early as possible in the product/feature cycle. This helps set the stage for future test planning discussions and most importantly, it gives your teammates an opportunity to call out any flaws in the strategy before you've done a bunch of tactical work such as creating individual test cases and developing test automation. Some key points I look for in this section are:

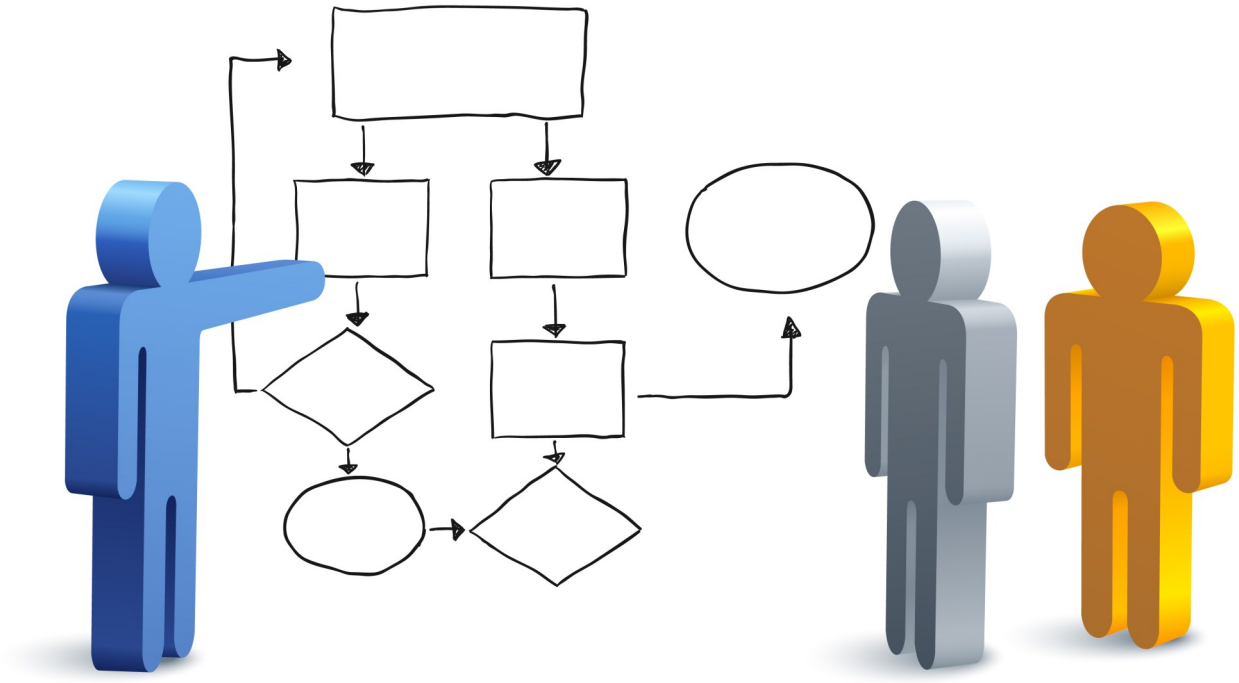
- An Understanding of the Feature Architecture - For the bits you are testing through "glass box" approaches, reflecting a simple understanding of the feature architecture in the test strategy section is very helpful. I don't expect to see a full copy of the feature architecture spec—just the "interesting" points called out and how the tester is using that knowledge to create an efficient testing strategy. For example, if the UI build is using MVC, a lot of testing can be done "under the hood" against the controller rather than driving all the tests via automation through the UI.
- Customer-Focused Testing Plan - Are you going to do any "real world" testing on your feature? Some examples include "dog-fooding" (either using the product yourself or deploying frequent drops to real customers to get feedback), testing with "real" data, install/uninstall testing on "dirty" machines (as opposed to cleanly installed VMs, for example).
- Test Case Generation Methodology - List the approach you're taking to generating test cases. Are you using the typical "sit around a table and brainstorm" approach? Model based testing? Pairwising? Automatic unit test stubbing tools?
- Identification of Who Will Test What - For example, developers will write and run unit tests before check-ins, testers will focus on system level, integration, and end-user scenarios, etc. This serves multiple purposes: reducing redundancy, driving accountability for quality across the feature area.

Testing Risks and Mitigation Plans

What known risks are you aware of? Some examples might be building on top of a new platform/framework that hasn't been fully tested yet, tight schedules, or relying on components delivered from external sources or other teams. Once you've called out the major risks, list what you are doing to mitigate the chances each one could impact your project? For example, do you have escalation paths identified in case you find bugs in those external components?

Testing Schedule

Ideally, this would be merged into a common team schedule so you can see dates from each discipline alongside each other. Dates/testing events listed should include dependencies (e.g. Test Plan Initial



Draft Complete" probably depends on "Feature Architecture Spec Complete" and "End User Scenario Spec Complete" dates being hit). Sharepoint is a great place to keep a team calendar for feature development, and the test plan can simply contain a link to the current schedule.

Test Case Details

Each test case listed should be written in such a way that someone who is only loosely familiar with the feature (and armed with the feature specifications) can execute the test cases. This is helpful for several reasons: churn among testers on the project, it facilitates hand-offs to test vendors or servicing teams (if you're lucky enough to have one), and most importantly, clarifying what is and isn't being tested by reducing ambiguity in the test plan.

- A clear description of the test case and steps
- A description of expected output/behavior
- Things to verify to prove the test case worked (or not, in regards to negative testing)
- "Clean up" steps, in a situation where the test changed something that might invalidate the entry point for other test cases
- Mapping back to an end-user scenario (this can be accomplished by grouping your tests by scenario, for example)
- Priority of the test case (from the end-user's perspective usually, though there are an infinite number of ways to prioritize)

And last but not least...

Make sure you use the right cover sheet! Seriously, if your team has a test plan template, by all means start with that! Chances are it's much more than a "TPS report." Templates give you a framework to help you think through your test planning process. You probably don't need to include each and every section in your final plan—just include the sections that are relevant to the feature/product you are testing.

There you go. I hope this helps you think through your own test planning process. ■

Jason Barile is the principal test manager for Visual Studio Team Foundation Server at Microsoft. He moved from Redmond to North Carolina in 2003 when Microsoft started the TFS project. Prior to TFS, he was a tester and test lead on the Windows User Experience team. Visit his blog at Testing Testing.

FEATURE ARTICLE

Mobile Application Testing: Process, Tools and Techniques

The outbreak of smartphones and tablets forces us to be digitally available with speed. Keeping pace with communication tool developments, **Lindiwe Vinson** defines the methods used at Organic, Inc. where she leads her team discovering bugs using various key programs for both PC and Mac platforms.

The market for mobile applications increases every day and is becoming more and more demanding as technology grows. In a new study, Yankee Group predicts a $4.2 billion "Mobile App Gold Rush" by 2013 which includes:

- Estimated number of smartphone users: 160 million
- Estimated number of smartphone app downloads: 7 billion
- Estimated revenue from smartphone app downloads: $4.2 billion

At Organic, our goal is to stay on the cutting edge of emerging platforms by launching new and diverse applications. We have this goal in mind when developing mobile web applications. We utilize some of the same styles of programming used for the developing of web applications. We also follow the same testing methodology employed for web development testing when testing our mobile applications.

1. Test Strategy is a high level document that defines "Testing Approach" to achieve testing objectives. The Test Strategy document is a static document meaning that it is not frequently updated. Components of the document include approach, risks, contingencies and recommendations, testing responsibility matrix, defect management process and resource requirements (schedule, tools, roles and responsibilities).

2. Performance Test Plan specifies how performance testing will proceed from a business perspective and technical perspective. At a minimum, a performance testing plan addresses dependencies and baseline assumptions, pre-performance testing actions, performance testing approach and performance testing activities.

3. Test Design Specification outlines, defines and details the approach taken to perform mobile application testing. The objective is to identify user flows and annotations, features to be tested, test scenarios, acceptance and release criteria.

4. Test Cases are derived from test scenarios and are identified in the test design specification. They are a set of test actions, test data/user input data, execution conditions, and expected results developed to verify successful and acceptable implementation of the application requirements.

5. Test Case Execution Summary Report provides information uncovered by the tests and is accomplished by the testing type. The report is used to relay the overall status of test execution on an iteration-by-iteration basis.

Although the mobile application testing process is basically the same we understand mobile devices have different peculiarities that must be kept in mind when deciding which testing types to use for authentication. The testing types used are predominantly unchanged but we do utilize different testing techniques and tools. Following are a list of testing types, techniques and tools used to support our mobile applications:

1. ADA Compliance Testing is used to measure and evaluate compliance to the Americans with Disabilities Act requirements. With mobile devices at an all-time high, there has been a surge of interest in developing applications that are in line with Mobile Web Best Practices (MWBP). To test accessibility we used the following tools and techniques.

    - Create a URL test harness. The URL is checked via W3C mobileOK Checker, a free W3C service that validates the level of mobile-friendliness.

    - The other test consists of using Apple's Assistive Technology to test for screen magnification and VoiceOver for the blind and visually impaired.

2. Automated Testing is achieved using an emulator and a performance testing tool. The test runs on the device itself and is controlled by the PC. Results are captured using the performance testing tool. More details are provided below in the Performance Testing section.

    - eggPlant is a QA automation and software testing product that allows you to emulate mobile devices and automate the testing. EggPlant can be downloaded for the Windows or Mac platforms.

3. Database Testing is very important for all applications. We check for data integrity and errors while editing, deleting and modifying the forms and all other DB related functionality. This testing is done manually, without the use of any testing tools.

Key Tools for Mobile Testing Needs:

Automated Testing

⇒eggPlant

Compatibility Testing

⇒iPhoney
⇒iPad Peek
⇒Adobe Device Central CS5
⇒DeviceAnywhere™

Mobile Analytics Testing

⇒Flurry™
⇒Charles Web Debugging Proxy
⇒Empirix eLoad Expert

4. Compatibility Testing assures the application works as intended with the selected device, operating system, screen size, display and internal hardware. Following are a list of tools that simulate different devices, operating systems, screens, etc.:

   a. iPhoney is a free iPhone simulator powered by Safari (used on a MAC OS platform only).

   b. Pad Peek allows you to see how your websites look when rendered on the iPad. This simulator is also free.

   c. Adobe Device Central CS5 allows you to plan, preview, and test and deliver mobile applications. It is available with the Adobe Creative Suite® editions: Photoshop, Illustrator, Flash Professional, Dreamweaver After Effects and Fireworks.

   d. DeviceAnywhere™ allows you to compose automated tests that run across multiple devices and multiple platforms/OS's. DeviceAnywhere™ is a paid solution providing monthly and/or hourly options.

5. Functionality Testing includes the testing of controls, storage media handling options, and other operational aspects. Functionality testing for the mobile application is black-box testing and assures that the application functions per the business specifications. This testing is done manually.

6. Interoperability Testing includes testing of different functionalities within the iPad. For instance we uncovered that iTunes and Pandora end the play of music when launching the BroadFeed™. Interoperability testing had uncovered a major defect.

7. Mobile Analytics Testing is one of the most important tests and validates our ROI. We used Flurry™ to collect the analytics for BroadFeed™. To test correct implementation of analytics, we verified page and link tags, redirects, page source and user attributes as well as data capture.

   a. Used Charles Web Debugging Proxy to verify the page and link tags, redirects requirements. This was achieved by changing the proxy settings in Charles then on the iPad; changed the Wi-Fi settings; "HTTP Proxy", selected the Manual

button and entered the desktop's IP address.

b. Used the Flurry™ Dashboard to validate the data was captured correctly. The dashboard view provided us with snapshot of user metrics and usage.

c. Performance Testing is used to load and stress test the mobile application and database servers. To conduct performance testing we first created a test harness. Once this was created, we used Empirix eTester to record the script used to preform load and stress testing. Empirix eLoad Expert allowed us to easily and accurately test the performance and scalability of BroadFeed™ to ensure our customers would have the best possible experience. ELoad Expert simulated concurrent users, which allowed us to analyze the performance and identify any potential database issues.

8. Power Consumption Testing uncovers defects related to battery drainage caused by the application. Device settings can drain the battery life and this makes it hard to determine if the mobile application or the settings are the cause. The following are devices and the different testing methods for testing power consumption:

   a. IPhone, iPod & iPad settings are adjusted; screen brightness, minimize use of location services, turn off push notifications, turn off other downloaded applications, fetch new data less frequently and turn off push mail. Then run the mobile application to determine the rate it took for the battery life to decrease. This testing is done manually without any testing tools.

   b. Nokia Energy profiler is a stand-alone test and measurement application which allows monitoring the battery consumption on a target device.

9. Usability Testing is used to verify mobile interface, navigation, and intuitiveness of the application, as well as consistency, and soberness of color schemes.

A list of mobile device emulators used for testing:

Android emulator
Blackberry emulator
Dot Mobi emulator
Firefox Mobile emulator
Klondike WML emulator
LG emulator
Microsoft Devices emulator
Motorola emulator
Mozilla Fennec emulator
NetFront emulator
Nokia emulator
Opera Mini emulator
Opera Mobile emulator
Palm Pre / iPhone emulator
Samsung Java emulator
Samsung Platform emulator
Windows Mobile emulator

What we know for sure is that there will always be some level of manual testing when launching new applications, whether it is web or mobile. The solutions we use for testing combine manual testing, remote-manual testing, and a lot of testing using emulators and performance testing. We accomplish our testing goals utilizing an array of testing types to support the different techniques. We combine testing tools to help with the validation process. We try to remain cost effective by using freeware and in-house tools which allows us to conduct testing quickly and efficiently. ■

Lindiwe Vinson is the Director of Technology at digital marketing agency Organic, Inc. and an expert in Quality Assurance. With more than twelve years of diverse experience, Lindiwe has a strong working knowledge SDLC, test planning, resourcing and contingency planning, metrics and defect reporting and performance testing; which includes an in-depth knowledge of engineering, testing, process management, change management and configuration management. She frequently contributes to Threeminds, Organic's digital marketing blog. For more information, visit Organic.

RELATED ARTICLE

User Stories, Scenarios & Use Cases

Distinguishing these terms from each other can be rather confusing. In an attempt to go back to the basics, **Nadine Schaeffer** explains in detail the benefits and the necessity of using realistic situations.

When designing and developing software and sites, I hear these three terms–user stories, scenarios and use cases–employed all the time, often interchangeably. While I am not intending to issue a didactic nomenclature smack-down, some definition and clarity can't hurt. Most importantly, when people are working together on a project team and trying to create and agree to any of these three kinds of deliverables, consensus goes a long way.

In general, all three terms represent ways that application teams try to define and understand what the app will do from the user perspective. User stories, scenarios and use cases can therefore be understood as framing devices; they provide mechanisms to understand and plan technology development around the end user, instead of focusing on development and product features around the constraints of the business, the platform or the underlying development of languages and libraries.

The exact purpose and audience for each term varies. Scenarios are generally used by user research people to communicate with design teams. User stories are written largely by project/product managers as part of definition and requirements documentation. The audience for use cases is primarily developers who need to write test cases and need to understand if their data objects are handling the appropriate i/o, errors, and exceptions. Now let's examine closely all three terms, associated deliverables and processes.

Scenarios

Let's start with scenarios. They are usually tied to personas and are part of creating a story about who the user of a particular technology is, what they want, what they know. A scenario is therefore usually written in narrative form, perhaps with pictures and illustrations as well. Scenarios are generally written at the beginning of a project during discovery and requirement gathering phases.

They provide human-centered anchors to guide design and development by providing tangible faces, names and stories for how the technology will be used.

A common generic format for scenarios goes something like this:

SCENARIO FORMAT

A person by the name of ***<John Doe>*** is ***<xx>*** years old and lives in a mid-size well-known city. ***<John Doe>*** works for a big bank, and has ***<this level of education>***. ***<John Doe>*** wants to ***<accomplish a task>*** because of ***<these motivations>.***

User Stories

Now what about user stories? User stories are generally used by Agile development teams. They are meant to replace long complex documentation with short sentences that capture the essence of a user's needs.

They are intentionally short and granular—each story captures just one task or action. User stories are defined during development before or at the beginning of each sprint.

User stories are generally written in this kind of syntax:

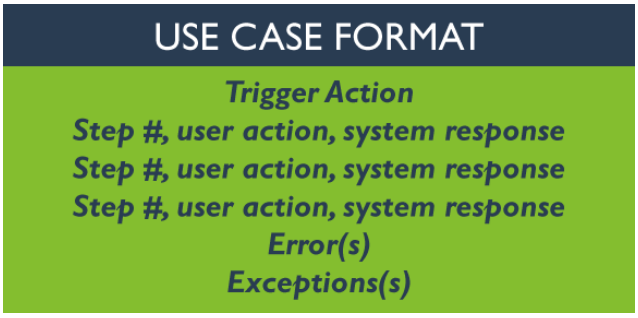
USER STORY FORMAT

A ***<type of user>*** wants to ***<do something>*** so that ***<this thing happens>***.

Use Cases

Finally, let's describe use cases. Use cases are generally written as part of detailed product requirement documentation. They capture the goal of an action, the trigger event that starts a process, and then describe each step of the process including inputs, outputs, errors and exceptions. Use cases are often written in the form of an actor or user performing an action followed by the expected system response and alternative outcomes.

Use cases tend to employ this kind of format and content:



Still confused? Maybe a more tangible real-life example will help. . .

Scenario:
Josh is a 30 something mid-level manager for an ad agency, metro-sexual and beer aficionado. He likes to try new and exotic beers in trendy locations. He also enjoys using a variety of social apps on his smart phone. He reads a review on Yelp of a new burger & beer joint downtown with over 100 beers on tap, and decides to go walk over after work and check it out.

User Story:
A user must want to find a bar and drink a beer.

Use Case:
Customer walks to the restaurant
Customer enters the restaurant
Customer finds a seat at the bar
Customer scans the menu
Customer selects a beer
Customer orders selected beer
Bartender takes order
Bartender pours beer
Bartender delivers beer
User drinks beer
User pays for beer

In Summary
Use cases, stories and scenarios are not interchangeable terms–each defines a specific process and set of deliverables. That being said, there are no hard and fast rules. It makes sense to review the needs of your project, what stage of development you are in, the skills and familiarity levels of the team, and then choose the user and task definition process that will work best for you.

Links & More Info

- This is a good example of a well written user scenario Accessibility is often over-looked in user scenarios, but not in this excellent example.
- This PDF includes more definition for user scenarios as well as an example.
- Here is Wikipedia's take on scenarios.
- Here is a good summation article about user stories.
- Good examples and instructions for writing great user stories.
- You can buy a whole book about writing great user stories Wikipedia of course has an overview page as well about user stories.
- How to write user stories using cards.
- Wikipedia has a good definition of use cases.
- And here is a good example of a use case about using an ATM.
- A good article on forming good use cases.
- Another longer article on writing use cases (with stick figures).
- A travel related set of use cases.
- A very usable template for use cases. ■

Nadine Schaeffer is a consultant professional with over fifteen years experience planning and building internet, mobile and desktop applications. She has aided many companies large and small launch successful technology and branding projects. In the past years, her work has focused on building successful user interface designs and flexible information architecture systems that integrate web 2.0 technologies as well as social and community initiatives. For more information, visit CloudForest.

SPOTLIGHT INTERVIEW

Janet Gregory

Janet Gregory draws from her own experience in helping agile teams address alternative ways to cope with road-blocks including projects without clear documentation, testers with limited domain knowledge and dealing with either black box or white box testing.

For testing on projects without clear documentation, is exploratory the only method? I often make "tester errors" on these projects. I might report wrong defects, or miss errors because I do not fully understand what to test. Have you come across this and if so, what methods would you recommend?

Unfortunately, many teams assume that the story is the only documentation they get and try to make the best of what they have, causing many misunderstandings to occur. A one sentence story is only one part of what is needed. Examples defining the expected behavior, plus one or two misbehaviors (what happens if), adds a lot of clarification.

However, it is the third piece that is absolutely necessary—the conversation that happens around the story and the examples (acceptance tests). If it is a complex story, you may even add a few extra examples to show alternative paths. I find that if I need too many examples to show the intent of the story, it likely should be broken up into more than one.

In the iteration planning session, I will start thinking about variations I need to test, and will write them down as they are discussed. These variations become the documentation as I automate them. I give these tests to the developer(s) who are coding the story, and we can collaborate even more to make sure we have a shared common understanding.

My exploratory testing can then concentrate on what I hadn't thought about, or about risk areas that have been identified. If there are no serious issues, what I usually find, feeds into new stories or better understanding of the application.

How would you measure productivity and quality of an agile test team?

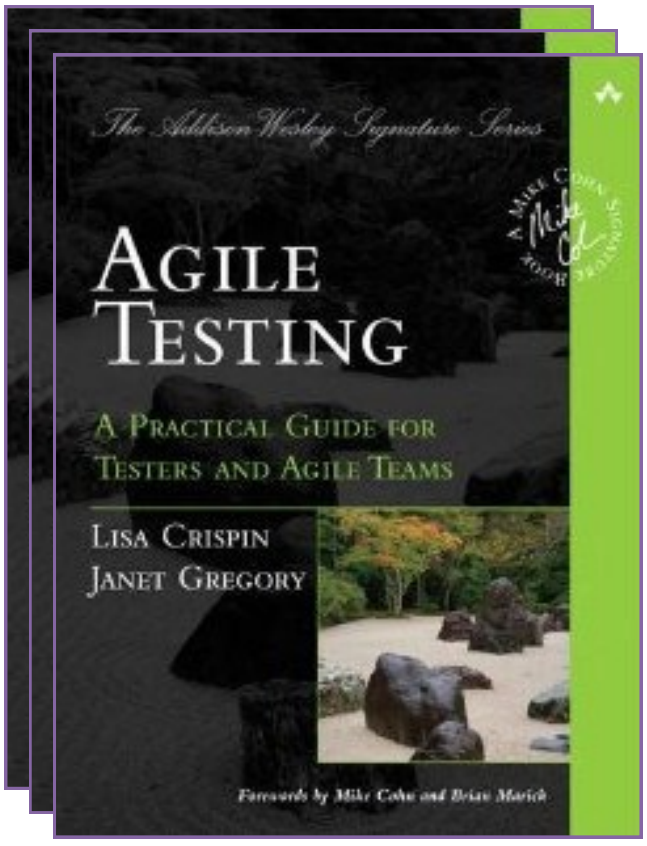
First, I'm not sure I'd measure productivity except maybe velocity, and that isn't comparable between teams. Absolute numbers related to productivity is really difficult to capture. I would rather see teams monitor their trends—is their velocity increasing on average? Or decreasing?

One of the issues I hear the most is about interference or blocks from other teams. When you see a problem, measure it. For example, capturing how much time is spent on non-project specific tasks against the actual velocity of the team may capture a correlation between the two, so, the problem can be addressed.

The simplest measure of quality that I have used is "How many defects or issues are reported in production by our customers?" If we keep our internal development process clean and aim for zero known defects escaping each iteration, we hope that very few defects make it to production, and the number reported is low. Of course, if the customer isn't using the system, this metric is irrelevant.

In a hypothetical worst-case scenario, an offshore team without exposure to the customer, possessing limited domain knowledge and receiving less-than-adequate requirements documents, what methods can test teams use?

This question can be taken one of two ways: either the developers are off-shore, or the testers are off-shore. I am assuming it is the developers who are off-shore and delivering code to testers after it is completed. There are a few things the tester and business users can do to help this situation. Ideally, some of the team from one location or the other have met face to face so they start building a relationship. In this worst case scenario, I'm guessing that didn't happen.



First, I would suggest practicing ATDD (acceptance test driven development). Instead of handing over a story, hoping the programmers understand the nuances and make the right decisions, help them by giving them the tests you are going to run along with the story. The tests become documentation that help drive development.

The trick is to keep the tests generic enough that you are not telling them how to code, but instead, what needs to be delivered. Let's use a simple example: If you are creating a login and password page, your acceptance tests may look something like this if written in a BDD (Behavioral) manner. [Note: validation rules are defined in wiki or someplace.]

Acceptance Tests (high level by customer)

- Expected Behavior: Given that I am an existing user, I enter the login page and enter a valid user name and password, then I am directed to the appropriate subsequent screen.

- Misbehavior: Given I am an existing user, I enter an invalid user name or password, I get an error message "Incorrect User Name or Password," and then I am presented with blank fields to reenter the information.

Extended tests delivered to the programmer might be a simple list including combinations of valid user names and passwords and invalid ones depending on the defined validation rules. Some examples might be:

- Blank user name, valid password – error
- Valid user name, blank password – error
- Duplicate user names – error
- Duplicate passwords – valid
- Trailing blanks – valid (expect developers to trim the trailing blanks)
- Leading blanks – (do we want them trimmed or rejected?) worth asking the question

As testers, we want to help the programmers succeed.

In Data Warehousing or Business Intelligence (BI) projects, could we apply Agile? What are common difficulties while testing these projects? Any best test methods for these?

One of the most common difficulties in testing data warehousing projects is breaking up the features into testable chunks.

When my team is cemented into "doing what we always do," how can I introduce new test methods?

One of the most powerful examples I have to share is where the QA Director introduced weekly lunch and learns. . . . Each week, she had one of the team members give a presentation on a new idea or introduce a new tool. It encouraged research by individuals to reach beyond what they already knew, gave opportunities for others to follow-up and try new things, and introduced the idea of sharing ideas and learning from each other.

Teams sometimes need to shift their mindsets quite a bit to accomplish this.

Successful agile BI teams have realized that they can test ETLs individually ensuring row counts, totals, column widths and data types, where appropriate, remained consistent as the data moved through the ETL stream. These are areas that are suited for automation and can become part of the regression tests suite.

Other areas like consistency or accuracy of data quality cannot be automated and need to be addressed with tools like exploratory testing.

One of the difficulties can be getting feedback early from the customer. Is the data meeting their needs? The quality of the data is what becomes important in BI projects.

In general, how much do people employ user scenario testing as opposed to requirements validation and exploratory testing methods?

I'm not sure I'd differentiate the testing in that way. I don't think that user scenario testing is "opposite" to either requirements validation (which is testing that the code is doing what we thought it should), or exploratory testing (which is more like testing our expectations). Scenario testing is checking that the workflows work which could be considered in either.

I think it is important to understand why you are testing, and not get hung up on the type you are doing. You might want to investigate the agile testing quadrants (several chapters in our book, or else Brian Marick's blog http://www.exampler.com/old-blog/2003/08/21/#agile-testing-project-1 .

Features are broken up into multiple stories. I like to define acceptance tests at the feature level as well as the story level. These feature level tests usually look a lot like user workflows or scenarios. This makes it a different level of test.

When I practice ATDD, I define my tests up front (testing what I can think of about the story - requirements), and collaborate with the programmers so we have a shared common understanding of what we are building. I have many tools I use to determine what tests I need. When I do my exploratory testing, I engage other skills, some which might include using my domain knowledge for testing various workflows.

Are there companies you're aware of that use older test methods like cause-effect graphing or model-based testing (from black box, not white box)?

Cause-effect graphing is one method to determine what to test; there are many others methods as well —some are used more than others. I think testers use different tools to help them get adequate coverage and shouldn't limit them to one type. Each feature/story may want you to try something different.

Model-based testing is an automation model to allow various flows through the system. I do know of organizations that have tried to model their legacy system for their automation rather than try to get complete coverage using static data. They have had different levels of success.

In most organizations that are doing on-going automation on each user story, they define their automation tests as they go, so I'm not seeing the model based testing as much.

With test and lifecycle management tools so common today dictating test design instead of test teams developing a design on their own, do you see test teams being limited by tool use?

It is easy to rely on tools to do our work for us, but no tool can think like the human brain. I would hope that testers see the limitations of the tool and challenge their own perceptions. One area where a tool cannot compete is in the area of exploratory testing.

When my team is cemented into "doing what we always do," how can I introduce new test methods?

The first step is recognizing you are in a rut, which you have already done. Now, you need to get the rest of the team to want to try new ideas. One of the most powerful examples I have to share is where the QA Director in one organization I worked with, introduced weekly lunch and learns. She created a learning organization within her team.

Each week, she had one of the team members give a presentation on a new idea or introduce a new tool. It encouraged research by individuals to reach beyond what they already knew, gave opportunities for others to follow-up and try new things, and introduced the idea of sharing ideas and learning from each other. ■

An agile testing coach and practitioner, Janet Gregory is the co-author of Agile Testing: A Practical Guide for Testers and Agile Teams *and a contributor to* 97 Things Every Programmer Should Know. *Janet specializes in showing agile teams how testers can add value in areas beyond critiquing the product; for example, by guiding development with business-facing tests. For the past ten years, Janet has been working with teams to transition to agile development, and teaches agile testing courses and tutorials worldwide. She enjoys sharing her experiences at conferences and user group meetings around the world, and was named one of the 13 Women of Influence in testing by* Software Test & Performance *magazine.*
For more about Janet's work, visit www.janetgregory.ca or visit her blog at janetgregory.blogspot.com.

2010 GLOBAL TESTING SURVEY RESULTS

Test Methods, Tools & Metrics

Data was compiled and analyzed by Michael Hackett, LogiGear Senior Vice President. This is the fifth analysis of the 2010 Global Testing Survey Series. More survey results will be included in subsequent magazine issues. To read past analysis, visit http://www.logigear.com/survey-response-overview.html.

I. METHODS

M1. The test cases for your effort are based primarily on:

	Response percent	Response count
Requirements documents	61.3%	46
Discussions with users on expected use	2.7%	2
Discussions with product, business analysts, and marketing represent-atives	9.3%	7
Technical documents	4%	3
Discussions with developers	8%	6
My experience and subject or technical expertise	12%	9
No pre-writing of test cases, I test against how the application is built once I see it	2.7%	2
Guess work	0%	0

Result analysis: This confirms conventional wisdom. Over 60% of the respondents list requirements documents as the basis of their test cases. This brings up two important discussion points: 1) test cases can only be as good as or as thorough as the re-quirements and 2) past survey results exhibit that most testers are hired because of their subject matter expertise. This sub-

ject matter expertise is the primary basis for test case development for a far-distant 12%.

Some test teams complain regularly about requirements documents they receive. I would assume that reliance on subject matter expertise would have been a more common basis for test case development.

M2. Are you executing any application-level security and/or usability testing?

	Response percent	Response count
Yes for both	42.1%	32
Yes for usability	23.7%	18
No for both	23.7%	18
Yes for security	10.5%	8

M3. Are you conducting any performance and scalability testing?

	Response percent	Response count
Yes	74%	54
No	26%	19

M4. Does your group normally test for memory leaks?

	Response percent	Response count
No	52.1%	38
Yes	47.9%	35

M5. Does your group normally test for buffer overflows?

	Response percent	Response count
No	56%	42
Yes	44%	33

M6. Does your group normally test for multi-threading issues?

	Response percent	Response count
No	50.7%	38
Yes	49.3%	37

M7. Do you do any API testing?

	Response percent	Response count
Yes	62.7%	47
No	37.3%	28

Result analysis M2- M7: As test engineers, you need to know what types of testing need to be executed against your system even if you are not the person or team executing the tests. For example, performance tests are often executed by a different team separate from those that execute functional tests. However, your knowledge of test design and data design will help those executing these tests.

Knowledge in other tests can also help cut redundancy and shorten a test cycle. Most importantly, it becomes a serious problem if you are not executing these tests because you don't know how or hope someone else will take over. If you are not doing memory tests because you think or hope the developers are, this is also a mistake.

API testing can find bugs earlier in the development cycle and has easier defect isolation. If API testing is not being practiced, you should have a good reason.

M8. What percentage of testing is exploratory/ad hoc and not documented?

	Response percent	Response count
10 - 33% (important part of our strategy)	46.7%	35
Less than 10% (very little)	26.7%	20
33 - 66% (very important, approximately half, more than half)	13.3%	10
66% - 99% (our most important test method)	10.7%	8
0% (none)	2.7%	2
100% (all our testing is exploratory)	0%	0

Result analysis: With almost half responding that 10 -33% of exploratory testing plays an important part of their strategy, my biggest concern is that the project team knows and understands your use of exploratory testing and the difficulty in measuring it.

The high percentage calling it important and the difficulty measuring exploratory testing often leads to incorrectly sizing a test project or increasing risk in cutting schedule time allotted for exploratory testing.

I expected a bigger reliance on exploratory testing with only over a quarter responded, "less than 10 %." Most team teams say they find most of their bugs running exploratory tests as opposed to validation test cases. This may still be true, but many test teams may lack the time to do exploratory tests.

M9. How effective is your exploratory/ad hoc testing?

	Response percent	Response count
Somewhat effective, it is useful	54.8%	40
Very effective, it is our best bug finding method	39.7%	29
Not effective, it is a waste of time/ money testing	5.5%	4

Result analysis: Almost 40% said their exploratory/ad hoc testing very effective and the best bug finding method. I agree and hope you communicate to your teams how effective it is and your reliance on it.

M10. How is exploratory/ad hoc testing viewed by project management?

	Response percent	Response count
Somewhat effective, it is useful	58.6%	41
Essential, necessary for a quality release	20%	14
Very effective, it is our best bug finding method	11.4%	8
Not effective, it is a waste of time/ money testing	10%	7

Result analysis: Close to 60 % of respondents say management views the strategy as somewhat effective. In the previous questions, nearly the same percentage saw the testing as useful.

This surprises me. Very often testers see ad hoc testing as more valuable than management who often see it as immeasurable and unmanageable.

M11. What is the goal of testing (test strategy) for your product?

	Response percent	Response count
Validate requirements	34.20%	25
Find bugs	26%	19
Improve customer satisfaction	23.30%	17
Cut customer support costs/help desk calls	8.20%	6
Maximize code level coverage	5.50%	4
Improve usability	1.40%	1
Improve performance	1.40%	1
Comply with regulations	0%	0
Test component interoperability	0%	0

Result analysis: The number one answer for what is the goal of testing is validating requirements. This is a surprise. Typically, finding bugs is seen by testers as the goal and management, business analysts or marketing see validating requirements as the main goal and job of test teams.

Even with this, about half the respondents see finding bugs and improving customer satisfaction as the goal of testing. We see a few times in this survey a large reliance on requirements as the basis of testing. This can be a problem with anything less than *great* requirements!

2010 Global Testing Survey Results Available Online:

Overview

Agile

Automation

Test Process & SDLC

M12. Which test methods and test design techniques does your group use in developing test cases? (You can choose more than one answer.)

	Response percent	Response count
Requirements-based testing	93.20%	69
Regression testing	78.40%	58
Exploratory/ AdHoc testing	68.90%	51
Scenario-based testing	56.80%	42
Data driven testing	40.50%	30
Equivalent class partitioning and boundary value analysis	27%	20
Forced error testing	25.70%	19
Keyword/Action-based testing	17.60%	13
Path testing	16.20%	12
Cause-effect graphing	12.20%	9
Model-based testing	10.80%	8
Attack-based testing	9.50%	7

M13. How do you do regression testing?

	Response percent	Response count
Both	47.30%	35
Manual	33.80%	25
Automated	14.90%	11

Result analysis: For 33% of respondents, regression testing is purely manual. I see this commonly in development teams. There are so many good test automation tools on the market today that can be used on more platforms that teams not automating their regression tests ought to re-examine test automation. For all testers, test automation is a core job skill.

M14. Is your test environment maintained and controlled so that it contributes to your test effort?

	Response percent	Response count
Yes	83.1%	49
No	16.9%	10

M15. Are there testing or quality problems related to the environments with which you test?

	Response percent	Response count
Yes	64.4%	47
No	35.6%	26

M16. Are there testing or quality problems related to the data with which you test?

	Response percent	Response count
Yes	63%	46
No	37%	27

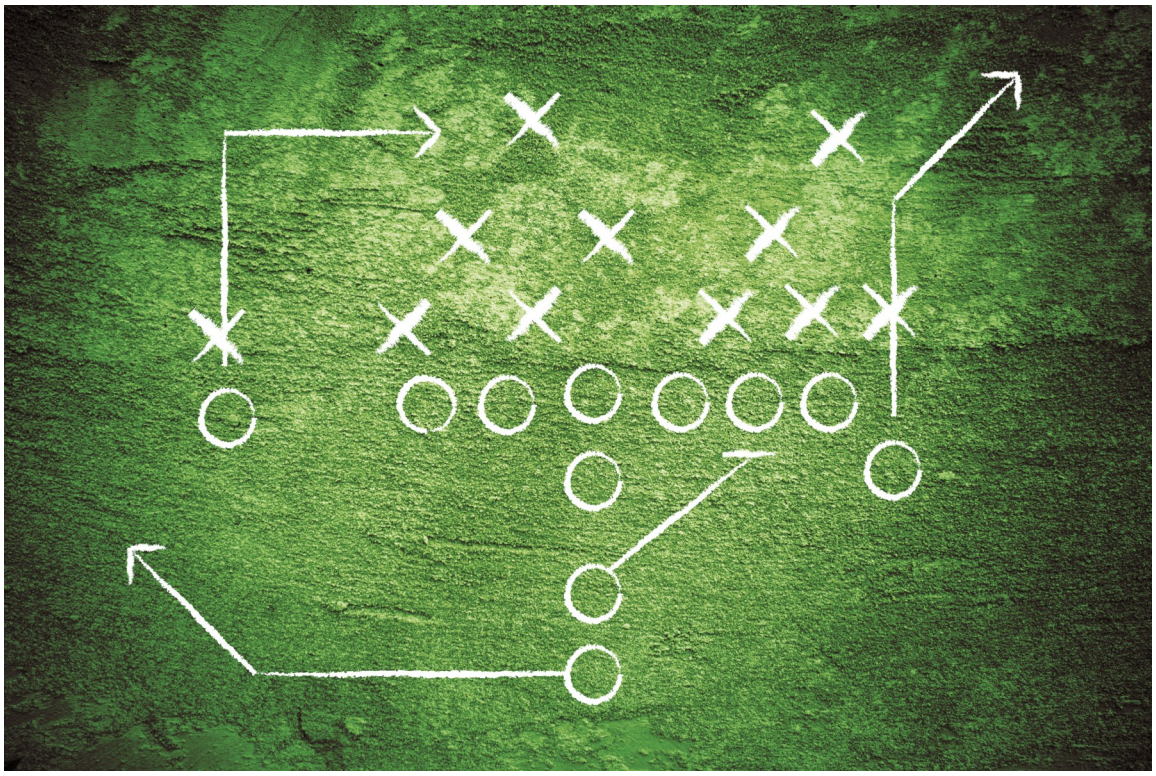
Result analysis M14- M16: Controlling test environments and test data is essential. Environments and data leading to testing problems is very common and very problematic! Building and maintaining great test environments and test data needs time and investment.

These answers confirm what I see in companies regularly—problems in environments and data not getting resolved. With some time and perseverance, fixing these would greatly improve the effectiveness of testing and the trust of the test effort.

II. TOOLS

T1. What testing-support tools do you use? (Please check all that apply.)

	Response percent	Response count
Bug tracking/issue tracking/ defect tracking	87.70%	64
Source control	54.80%	40
Automation tool interface (to manage and run, not write automated tests)	52.10%	38
Test case manager	50.70%	37
Change request/change management/change control system	47.90%	35
A full ALM (application lifecycle management) suite	19.20%	14



Result analysis: Thirteen percent do not use a bug tracking tool. This does not surprise me but it does many others that many test teams do not track their bugs!

About half of the respondents use a test case manager, and the same percentage uses a requirements manager or change control system. Half use a automation tool interface; these tools most commonly contain manual and automated test cases. Yet, only 20% use a full lifecycle ALM tool. A few years ago this number would have been much smaller. With each passing year—especially as more teams go agile or offshore work—this number will dramatically increase.

T2. I would describe our bug (bug, issue, defects) tracking as:

	Response percent	Response count
Effective	37.70%	26
Very effective; has a positive impact on product quality	34.80%	24
Adequate	20.30%	14
A mess	4.30%	3
Poor	2.90%	2
Hurts product quality/has a negative impact on product quality	0%	0

T3. What type bug tracking tool do you use?

	Response percent	Response count
Relational database tool (Bugzilla, TrackGear, TeamTrack, Team Test, ClearQuest, Homebuilt web-based or client server database)	68.10%	47
ALM tool that includes defect tracking	18.80%	13
Excel	8.70%	6
Email	2.90%	2
We do not track bugs	1.40%	1

Result analysis: This is a very positive move in our profession. Just a few years ago the number of teams using excel to track issues was significantly higher.

Excel is not an adequate issue tracking tool to sort, query, retrieve old issues from past releases, or control and manage access. With so many good and open source tools, there is no reason to be using a naive system.

T4. How many bug tracking systems do you use during a regular test project?

	Response percent	Response count
1	69.60%	48
Combination of tool and Excel and/or email	14.50%	10
2	11.60%	8
More than 2	4.30%	3

Result analysis: The problem of multiple bug tracking tools is common. In this survey, almost 30% of teams use more than one bug tracking tool. Most problematic is that almost 15% who use a tool, use excel and email. I see this often in my consulting work. It always causes headaches.

One team will not use another team's tool, developers have a work management tool and will not use the bug tracking tool so the test team has to use two tools, some remote team may not be allowed access to the internal tool so all their bugs get communicated in excel and email. It is a management problem, but it also lends to a more devious problem of giving the impression that testing is disorganized.

T5. How do you communicate and manage test cases?

	Response percent	Response count
A relational database/ repository focused on test case management (TCM, Silk Central, Rational Test Manager, TA, etc)	41.50%	27
Excel	21.50%	14
ALM tool that includes test case management	20%	13
Word	15.40%	10
We do not track, communicate or manage test cases	1.50%	1

Result analysis: The problem here is that almost 37% of teams are using MS-Word or Excel—that is dead data. It is difficult to share, edit, maintain, sort, query, and measure with these programs.

There are so many good test case management tools, some open source that make writing, editing/maintaining, sharing and measuring test cases so much easier. In my experience, there are very few good reasons not migrating to an easier and more sophisticated tool set.

There are also easy solutions to have test cases and bug tracking linked to the same tool. Test teams can graduate to a higher level of management, reporting and efficiency with tool sets.

T6. How are the test cases used? (Choose the MOST appropriate.)

	Response percent	Response count
They are used only for testers to execute tests	34.30%	24
They are used to measure and assess test coverage	30%	21
They are used to assess proper test execution	21.40%	15
They are used to measure and manage project progress	14.30%	10
They are not used during the project	0%	0

Result analysis: For teams not using test cases for more than execution, it may be useful to know that it is very common for them to have other usage.

T7. If you use a test case management tool, how is it used?

	Response percent	Response count
It is used to run our manual and automated tests	57.40%	31
It is used only for manual tests	40.70%	22
It is used to run only automated tests	1.90%	1

T8. If you have experience with success or failure regarding test tool use (ALM, bug tracking, test case management, automation tool interface, other) that is interesting or helpful to others, please write it below: (Comments from practitioners.)

1. "The best thing to do is manage the progress of the tests and see the bugs. You can measure the project's health."
2. "I find Bugzilla reporting and commenting adequate communication most of the time. Its only problem is when immediate problems surface - at that point an email to appropriate parties telling them to look at Bugzilla usually works. So does walking over to the developer and showing them the issue."
3. "So far Jira was the best bug tracking tool."
4. "If you want people to use a TCM or bug management tool, make sure it has good performance and it's simple."
5. "For a large project or program it is crucial to select a single method of tracking defects and what is considered defects versus 'issues.' This can lead to a great deal of confusion where defects identified as issues are not handled and addressed properly. I worked on a large project that various efforts had four different ways of tracking defects and issues. The result was that it was hard to assess the overall quality of the product that was being implemented."
6. "Testing should be driven by proven testing methodologies; not by the tool itself."
7. "Generating quality reports can be difficult using bug tracking systems."
8. Certain automation tools will not be suitable for certain type for projects."
9. "Test case management tools are not integrated to requirement management tools reason why our test cases are sometimes tested against obsolete functionality."
10. "Rally is very useful."
11. "Process discipline matters more than any tool."
12. "The tool is difficult to use for non-technical team members."

III. METRICS AND MEASUREMENTS

MM1. Do you have a metric or measurement dashboard built to report to your project team?

	Response percent	Response count
Yes	69%	49
No	31%	22

Result analysis: Anything worth doing is worth measuring. Why would almost 1/3 of teams not measure? Is the work not important or respected? Does the team not care about the work you do?

I am not measurement obsessed but when test groups do not report measurements back to the project team, it is very often the sign of a bigger problem.

MM2. If yes, are these metrics and measurements used or ignored?

	Response percent	Response count
This information along with schedule decides product release	43.40%	23
Some attention is paid to them	26.40%	14
This information decides product release	18.90%	10
Minimal attention is paid to them	5.70%	3
They are ignored	5.70%	3

Result analysis: It is good to see that over 60% of teams reporting metrics use the information to decide release. Test team metrics should help decide product release.

As with the previous question, if the project team is not paying attention to test measurements, that is often the sign of a problem.

MM3. What testing specific metrics do you currently collect and communicate? (You may select multiple answers.)

	Response percent	Response count
Test case progress	84.10%	53
Bug/defect/issue metrics (total # open, # closed, # high priority, etc.)	82.50%	52
Requirements coverage	49.20%	31
Defect density	34.90%	22
Defect aging	25.40%	16
Root cause analysis	25.40%	16
Code coverage	22.20%	14
Defect removal rates	22.20%	14
Requirements stability/ requirements churn	17.50%	11
Hours tested per build	11.10%	7

MM4. What methods/metrics do you use to evaluate the project status? (Comments from respondents.)

1. "Too many."
2. "Bug Rate Trends and Test Case Coverage (Feature and Full Regression)"
3. "Test case progress, defect status."
4. "Earned & Burned"
5. "Number and severity of remaining open bugs. 'Finger in the air' (tester intuition that some areas need more testing)."
6. "Executed test cases."
7. "Defect counts and severity along with test case completion."
8. "Bug/defect/issue metrics (total # open, # closed, # high priority, etc.)"
9. "Test case progress, defect density, and requirement coverage."
10. "Requirement coverage and test completed."
11. "Bugs found vs. fixed."
12. "Test coverage and defect metrics."
13. "Defects open along with test case execution progress."
14. "None."
15. "Are all the features 'tested.'"
16. "Track change proposals and outcomes (P/F) for all changes by project, by application, by developer, and by week."
17. "Exit criteria are agreed upon upfront, then metrics report progress against those. "
18. "Test case complete %, pass/fail %, # of high-severity bugs."
19. "Schedule deviation, defects detected at each stage of project."

MM5. Do you collect any metrics to evaluate the focus of the test effort, that is, to audit if you are running the right tests?

	Response percent	Response count
Yes	56.9%	37
No	43.1%	28

Result analysis: It is a step higher in responsibility and ownership when test teams evaluate their own work for effectiveness. Good work!

MM6. Do the metrics you use most help you:

	Response percent	Response count
Release better product	31.30%	20
Improve the development and test process	26.60%	17
Do more effective/efficient testing	23.40%	15
They do not help	18.80%	12

Result analysis: With 80% respondents using metrics to improve is great! More teams can be using metrics to point out problems, improve risk reporting, and give greater visibility into testing. Also, if your team is not capturing any measurements for the difficult reasons, it is safe to say your work is not respected, no one cares, or the team is purely schedule driven regardless of what testing finds.

I recommend you start a metrics program to improve your job skills!

MM7. Do you measure regression test effectiveness?

	Response percent	Response count
Yes	59.1%	39
No	40.9%	27

Result analysis: Regression test effectiveness is a growing issue in testing. Numerous teams have been doing large scale test automation for many years now. Regression suites can become large, complex or difficult to execute. Many of the regression tests may be old, out of date, or are no longer effective. Yet, teams are often afraid to reduce the amount of regression tests.

At the same time, running very large regression test suites can take up too much bandwidth and impede a project. If you are having problems with your regression tests, start investigating their effectiveness.

MM8. How much of an effort is made to trace requirements directly to your test cases and defects?

	Response percent	Response count
The test team enters requirements into a tool and traces test cases and bugs to them	41.50%	27
We write test cases and hope they cover requirements; there is no easy, effective way to measure requirements coverage	18.50%	12
The product/marketing team enters all requirements into a tool. Test cases and bugs are traced back to the requirements. Coverage is regularly reported	15.40%	10
We do not attempt to measure test coverage against anything	15.40%	10
We trace test cases in a methodical, measurable, effective way against code (components, modules or functions)	9.20%	6

Result analysis: Tracing requirements to a test case does not guarantee a good product. Measuring requirements coverage has become an obsession of some teams.

The big issue for teams doing this is that the test case can only be as good as the requirements. Gaps in the requirements, incomplete or even bad requirements will help assure a problem product. Tracing test cases to problematic requirements will do no one any good.

This practice can be really useful for measuring requirements churn, bug density, or root cause analysis. Tracing or mapping requirements can be a good method of assessing the relevance of your tests.

MM9. If code coverage tools are used on your product, what do they measure?

	Response percent	Response count
Code-level coverage is lines of code, statements, branches, methods, etc.	55.3%	21
Effectiveness of test cases (test cases mapped to chunks/ blocks/lines of code)	44.7%	17

MM10. How do you measure and report coverage to the team?

	Response percent	Response count
Test plan/test case coverage	45.30%	29
Requirements coverage	25%	16
We do not measure test coverage	15.60%	10
Code coverage	7.80%	5
Platform/environment coverage	6.30%	4
Data coverage	0%	0

Result analysis: *Coverage,* however you define this is crucial to report to the team. It is the communication of where we are testing. This is the crux of the discussion of what is enough testing. ■

VIET NAM SCOPE

Ca phe break in Ho Chi Minh City

Vietnamese coffee culture is the center of the social and business interactions among locals and foreign professionals.

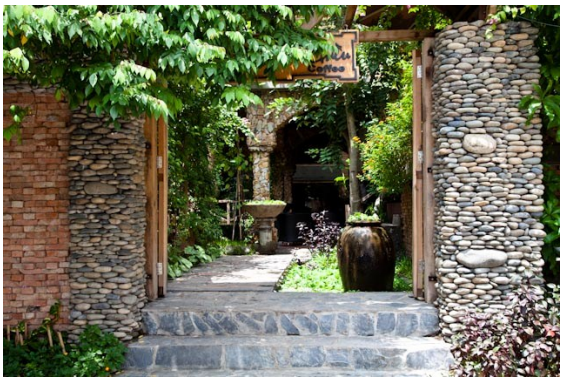
Lolita Guevarra

Amidst the populated sidewalks with countless motorbikes sprawling over the streets like water trying to run through rocks, Vietnamese cities are densely packed with chaotic movement. Yet, in all its overwhelming energy, the Vietnamese have definitely taken on the tradition of its once French colonists: drinking coffee.

Spelled "ca phe" and pronounced "café," Vietnamese locals can be seen sitting on pieces of cardboard on the sidewalk to high-end café parlors with dramatic garden landscapes and brick walls adorned with ivy. It is common for development teams or groups to take a ca phe break together for work and social discussions. Vietnamese professionals gather in groups scattered across the city spending their afternoons at cafes that it would be difficult for any visiting professional to not experience this coffee culture. The business is serious but the coffee is the enjoyment. The café culture in Viet Nam is one that has been overlooked but remains to be the epicenter of the people.

Second only to Brazil, Viet Nam follows as the largest exporter of coffee. The country mainly exports robusta coffee beans, a lesser quality to arabica which is considered the best containing richer flavors. Although coffee connoisseurs can turn their noses to Viet Nam's cheaper and less temperamental crop, they cannot however, dismiss the global market for instant coffee.

To those unfamiliar with Vietnamese coffee, it is known to have the density and feel of thick, black molasses. Its texture matches its strength in jolt. For the faint of heart, most foreigners order a *ca phe sua da*, which is coffee mixed with condensed sweet milk and ice. One sip evokes flavors recognizable in premium Belgium chocolate. The science to capture the sweet savor is not only the perfect blend of coffee and sweetened milk, but also the selection and the roast of beans.



The majority of coffee beans are grown in the central region of Viet Nam. Many tourists visit the central city Da Lat, enjoying the cool brisk air encompassing the city within the mountains. Lush and green, Da Lat has its fair share of sprawling fields lined with robusta, arabica, and mocha beans. Famous cafes in the area are run by long time café afficionados who have found the flawless union of mocha for sweetness and robusta for a strong finishing taste. Its unique flavor is attributed to its non-conventional roasting techniques where the beans are roasted slowly and in lower temperatures and normally added with butter and salt.

Either hot or chilled, with or without milk, the ca phe is at the core of Vietnamese culture. Rising with the sun, locals require their strong shot of ca phe, drinking an average of four cups a day. Women have also begun drinking ca phes, converging in four story cafes fully equipped with wall-rocked gardens and koi ponds linking one paradise to another with miniature Japanese bridges behind secluded walls. A country so abundant with enriched culture, its ca phe only adds to its unique experience. ■

LOGIGEAR MAGAZINE
JULY 2011 | VOL V | ISSUE 5

United States
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +1 650 572 1400
Fax +1 650 572 2822

Viet Nam, Da Nang
7th floor, Dana Book Building
76-78 Bach Dang
Hai Chau District
Tel: +84 511 3655 333
Fax: +84 511 3655 336

Viet Nam, Ho Chi Minh City
1A Phan Xich Long, Ward 2
Phu Nhuan District
Tel +84 8 3995 4072
Fax +84 8 3995 4076