DEDICATED TO SHOWCASING NEW TECHNOLOGY AND WORLD LEADERS IN SOFTWARE TESTING

# LogiGear MAGAZINE

## The Big Testing Issue

## Making BIG Testing A BIG Success

**By Hans Buwalda, CTO, LogiGear**

*Expanding Usability Testing to Evaluate Complex Systems*
**By Ginny Redish**

*Spotlight Interview: Salesforce.com*

*How to Write a Great Software Test Plan*
**By Robert Japenga, MicroTools**

April 2012 | Volume VI | Issue 2

# TestArchitect™

YOUR HIGH VOLUME TEST AUTOMATION SOLUTION

## TestArchitect™ features:

- All-In-One Solution: Test Management, Test Development and Test Automation
- Action Based Testing™ Methodology
- Built-In Customizable Automation
- Remote Test Execution
- Customizable Dashboard

# LETTER FROM THE EDITOR



## Worldwide Offices


**United States Headquarters**
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +01 650 572 1400
Fax +01 650 572 2822

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

**Viet Nam, Da Nang**
7th Floor, Dana Book building
76-78 Bach Dang
Hai Chau District
Tel +84 511 3655 33
Fax +84 511 3655 336

www.logigear.com
www.logigear.vn
www.logigearmagazine.com




Submission guidelines are located at http://www.logigear.com/magazine/issue/news/editorial-calendar-and-submission-guidelines/

Big and complex testing. What do these terms conjure up in your mind?

When we added this topic to the editorial calendar, I had the notion that we might illustrate some large or complex systems and explore some of the test and quality challenges they present. We might have an article on: building and testing the software for a rocket to Mars, and discuss the complex infrastructure behind it. (This, by the way, has been done many times, primarily to highlight the large scale system failures and huge sums of money wasted when projects of a massive scale are shortchanged on adequate planning, communication and testing). We could do the same for the air traffic control system's infrastructure, and a dozen other big software development projects that instantly come to mind. But big and complex are different to different people.

Big and complex software systems have been woven into our daily lives, and in many cases, those lives literally depend on them. It certainly, and justifiably, might give us pause when we consider that it's people like you and me who test these systems. Medical devices, online banking, missile guidance systems, prescription drug systems – it is a big list, and you and I *hope* they are tested well!

When I visit companies for consulting or training, I very often hear: "We have a *really* complex system! It's too difficult to diagram or describe." After one minute of hearing their explanations, I understand it to be a database with a web front end. Simple enough. But after five minutes, it's an inventory control system with tax and shipping integration with three varieties of credit card processing, all tied into reporting and accounting systems, accommodating three languages that all must work on five browsers and a variety of mobile devices. Indeed, what seemed simple enough at first, mushroomed in complexity very quickly. How do they test it? It has too many moving parts belonging to too many different groups. Each group has its own schedule, headaches and problems, and integrations of third party software.

I worked at Palm Computing during its early days. Palm was a pioneer in the handheld devices, smart phones and mobile computing systems that we all take for granted today. We thought *we* were complicated (at the time, we certainly were): changing hardware, changing OS, changing apps, *hotsynching* (synchronizing) to a wide variety of PCs, all in eight languages. Very ambitious indeed – and very complex testing. And yet, in retrospect, the complexity of what we were dealing with then at Palm pales in comparison to so many of the systems I see today.

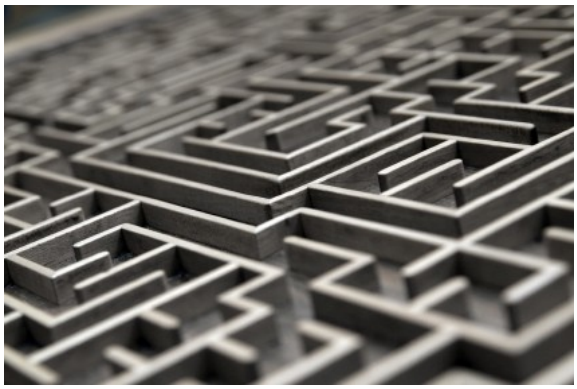
How is big or complex testing different than testing other-sized products? Maybe not so different after all: good test design, for example, is important no matter what size system you test! In this issue, we look at big testing from many perspectives, to examine both the differences as well as the fundamental constants of testing. LogiGear CTO Hans Buwalda provides us with a "big picture" look at complex systems; we see examples of complex system failures; Marcin Zręda reviews *Project Management of Complex and Embedded Systems*; Ginny Redish describes how thinking outside the box can lead to better testing; John Brøndum says the science of testing complex systems is constantly changing; I interview some Salesforce.com quality engineering directors about their approach in testing complex systems and I examine the professional characteristics of our global survey respondents. Finally, Robert Japenga shows us how to write a great software testing plan.

When I find myself in a distant country, late at night, hard up for cash, confronted with the ATM of a bank I've never heard of my plastic lifeline, somewhere deep inside its bowels while it awaits my bank's confirmation that I have funds available and I'm not totally indigent, my only thought is, "This had better work!" Who tests that and how well do they do it? Well, it may not always be possible to know *who*, but if you test big and complex bank or financial transaction systems, I hope we will give you some insight into *how*!

Michael Hackett
Senior Vice President
Editor in Chief

# IN THIS ISSUE





# IN THE NEWS

## Jane Fraser Video Interview on a 24/7 Complex System

At VISTACON 2011, Jane sat down with LogiGear Sr. VP, Michael Hackett, to discuss 24/7 complex systems.

As an industry veteran with more than fifteen years of experience, Jane Fraser brought her expertise from the e-commerce and telecom industry to the online gaming world when she joined Pogo in 2004. In her role as QA director, Jane oversees the QA department which includes Pogo.com, Club Pogo, Facebook games, iPhone games and a downloadable business. She has successfully launched more than 60 games in six territories including Scrabble and Battleship.

During her time with Pogo, Jane has provided leadership, established testing process, and managed a team of testers, which she has grown from six to a robust team of eighty in six countries.

*Her interview is available here: http://www.logigear.com/magazine/category/issue/videos/*

## LogiGear CTO, Hans Buwalda, to present at STAR*EAST*

This year's STAR*EAST* conference will take place from April 15—20 in Orlando, Florida where LogiGear's own Hans Buwalda will take the stage to talk about big testing.

STAR*EAST* is one of the premier software testing conferences where industry experts and peers in the test and QA community come together to create an open dialogue about the testing industry.

At the conference, Hans will present: *BIG Testing: Dealing with Large and Complex Testing Projects* and *Test Design for Automation.*

*To register or for more information on STAREAST, please visit: http://www.sqe.com/StarEast/*

## New LogiGear Magazine Website


It's been a long time coming but we've finally finished the migration to the new *LogiGear Magazine* website!

The new blog-style site, powered by Word-press, will provide a significantly improved user experience, making it easier to find and interact with content. Users can now comment on articles, explore our archives (which date back to 2005), and drill down into relevant categories such as agile, mobile and offshoring.

Anyone can also share articles easily with our social media links, access training resources and, using our new search func-tionality, view all the content from specific authors or topics.

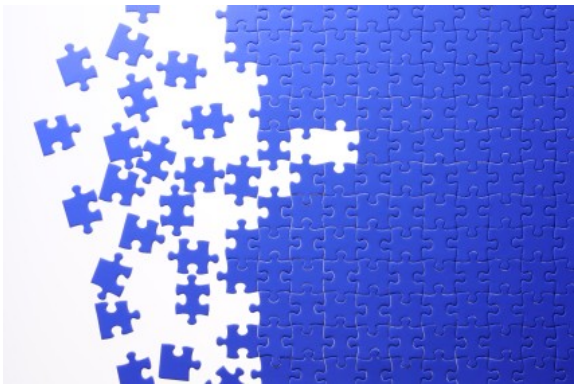
# BLOGGER OF THE MONTH

## John Brøndum

### More Just Isn't More...

Most have probably heard the expression 'less is more', or know of the 'keep it simple and stupid' principle. These are general and well-accepted principles for design and architecture in general, and something that any software architect should aspire to. Similarly, Richard P. Gabriel (a major figure in the world of Lisp programming language, accomplished poet, and currently an IBM Distinguished Engineer) coined the phrase 'Worse Is Better', in formulating a software concept that Christopher Alexander (a 'real' architect famous within software engineering) might have termed 'piecemeal growth' – e.g., start small and add later.

But what happens if we *continue* to add more, even if they are just small, simple pieces?



The suggestion is that more isn't just... *more*.

A bit of history digging reveals that, in 1962, Herbert Simon (granted the ACM Turing Award in 1972, and Nobel Memorial Prize in 1978) published *The Architecture of Complexity*, describing the structure and properties of complex systems. He concluded that complex systems are inherently *hierarchical* in structure, and exhibit near-decomposability property – e.g., a complex system is essentially a system of mostly independent systems. In 1972, P.W. Anderson (indirectly) builds this line of thought in "*More is Different*". Anderson argued that an understanding of how the parts (e.g., systems) work, does not equate to an understanding of how the whole works. We can all relate to the point that biology, medicine, and science can explain, to a large degree, how our bodies work, yet when all the parts are combined into a human body, elements such as emotion, intelligence, and personality form part of the 'whole', and are distinctly not medicine or biology. Yet, in Simon's terminology, the human body is a system of systems exhibiting the near-decomposability property – otherwise, heart transplant wouldn't be possible.

The near-decomposability is a property we desire as part of software design – although better known as *modularity*. Software architecture is basically 'to separate into components or basic elements', based primarily on the fundamental principle of *information hiding* first formulated in Parnas's seminal paper of 1972: *On the criteria to be used in decomposing systems into modules*. But as Richard Gabriel argued in his essay, *Design Beyond Human Abilities*, there is a key difference between software modularity, and the near-decomposability property of systems of systems.

Within a software engineering context, a module is traditionally defined by *what it does* rather than by *who produced it* – the latter is the definition used by Simon and Anderson.

This is a significant difference, and one we need to get used to as our software systems become increasingly complex.

Those of you who know Conway's law from 1968, shouldn't be surprised however. The 'law' states, "*a software system necessarily will show a congruence with the social structure of the organization that produced it*". In other words, the modular structure of a software system reflects that of the organisation's social structure – e.g., *who* rather than *what*. This doesn't imply that you should skip your 'structured design', 'object oriented programming', or 'patterns' course at uni and head down to the bar to socialise instead, but I think there is a number of implications that software architects, especially, need to pay attention to, when dealing with a system of *software* systems:

- The architecture will be at the mercy of the organisational structure or social network. Although I don't believe that software architects need to become experts in social science or organisational behaviours, it will be helpful to understand the basics. To get you started, I'd suggest a couple of books: *The Hidden Power of Social Networks: Understanding How Work Really Gets Done in Organizations* and, *Getting Things Done When You Are Not in Charge*

- Software architects can no longer pretend that they are software versions of 'real' building architects. We'll need more skills similar to those possessed by policy makers, negotiators, and other communicators (though not necessarily politicians). Robert Schaefer's '*Software maturity: design as a dark art*' is a place to start.

- The duplication of software functionality or applications within organisations or commercial enterprises isn't necessarily a bad thing in itself, but instead we need to focus on warning signs such as duplicated organisational roles (or responsibility overlaps), or lacking communication channels (related to the first point). I know many might be up in arms by the mere suggestion of duplication (wasting resources is never good!), but we need to be serious about the 'cost of re-use', versus the 'cost of (almost) re-implementation'. Even when viewed within the context of Total Cost of Ownership, my belief is that the former isn't always the clear winner.

- Focus on *interoperability* instead of *integration*. So what's the difference? NEHTA's Interoperability Framework captures the difference as the ability to maintain integration over time at minimum effort: the ability to maintain integration despite changes to, or replacement of, the communicating systems. Other references include the comprehensive '*A Survey of Interoperability Measurement*' – if you can't measure it, then you are not going to get it.

Today's realities of software development are essentially about adding, changing, or removing parts of an existing complex software system, through a continuous process of negotiations, bargaining, and policing. Our ability to do this is directly linked to our software's ability to interoperate. ■

**About John**

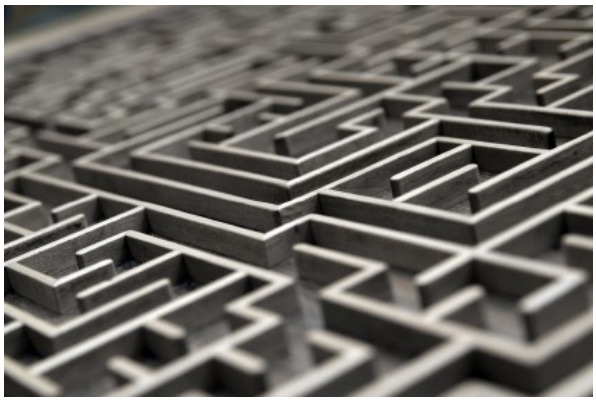
*John Brøndum has worked as an IT Architect and Consultant since 1997 for a number of technology companies, including IBM, across finance, retail, telecommunications, energy, and rail. With a wealth of project based experience (e.g., Service Oriented Architecture, Business Process Management, Reference Architecture), John is currently a Ph.D. candidate at UNSW and NICTA. His research is focused on the architectural complexities caused by increasing inter-dependencies between highly integrated enterprise applications. John works independently as a consulting architect, offering a range of architectural services.*
*Contact: mail@johnbrondum.com*

Cover Story

# Making BIG Testing a BIG Success

*There is no one recipe to make big testing a big success. It takes planning and careful execution of the various aspects, like test design, infrastructure and organization – a mix that can be different for each situation in which you may find yourself.*

**By Hans Buwalda, LogiGear**



In writing about big testing, the first question that comes up is, "What is 'big'?" In this article, "big" means, well, a big test.

Now, that does not necessarily mean a big system under test. There are, in fact, a number of factors that can make for a substantial testing effort. They include:

- Big volume of tests – for example, targeting a large system under test
- Concurrent variation of code branches
- Complex functionalities, needing many test situations or values
- Big variation in target platforms and configurations
- Big maintenance: rapid and frequent changes to the SUT
- Big or complex hardware infrastructure needed to support testing

"Big" is also a relative measure. It can mean different things for different organizations. For one company, 1000 test cases could be big, while another company may be looking at 1000 weeks of testing. Testing on ten machines at the same time can seem big, while another setup may involve a data center of ten acres. One definition of "big" could be that the size of the tests is not trivial: you need to think about how to organize and manage them. In other words: to make big testing a big success, it takes more than just making test cases.

In this overview I will describe three factors that I feel are key to big testing:

- Test design and automation
- Infrastructure
- Organization

### Test Design and Automation

When I have to deal with large or complex testing projects, I always look at overall test design first, since that is where the biggest potential gains lie. Organizing your tests can keep their sizes more manageable, and leads to a much better automation result. I will discuss automation first, then look at a way to best structure and design your tests.

For almost all big tests, at least the test execution will be automated. Automation is not a must: to be sure, human testing can have distinct advantages over automated tests. This is particularly the case if tests needn't be repeated much (For more on this, see the literature about exploratory testing).

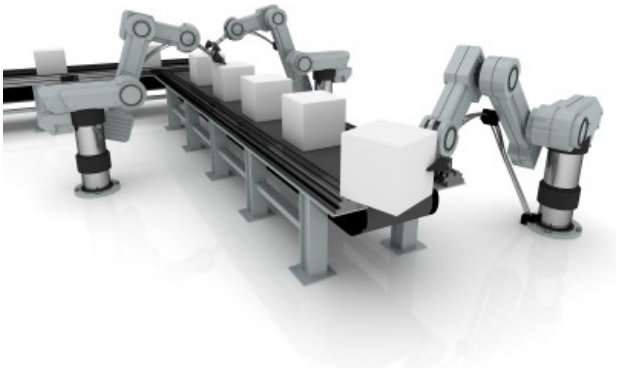
Several techniques have been developed for test automation. The three most common models are record & playback, scripting and keywords.

*Record & playback* can be useful to get to know your test tool and UI under test, and to quickly capture key steps in the automation. Note that this technique should not be used on its own for large scale test development; rather, recordings should be reused in functions or actions.

*Scripting* is often based on frameworks, like libraries developed in-house, or public ones like Selenium, a popular open source framework.

Scripting is an effective way to build automation, but difficult to scale up: the scripts need technical experience to build, and tend to become bulky and inaccessible pieces of software when scaled.

Of the three more common automation techniques, the *keywords* technique, in my experience, is the way to go for larger scale testing. For one thing, this system allows more people to get involved. By encapsulating elements of the automation in keywords, the technique offers a natural way to keep unneeded details hidden from a test design, which in turn keeps large tests more maintainable.



However, keywords alone do not constitute a method, and by themselves are not sufficient to achieve success. One method that builds on keywords is Action Based Testing (ABT). ABT places the focus on how tests can best be organized and designed to accommodate their effective automation. In ABT, tests are kept as a series of *actions* in a *test module*. A test module looks essentially like a spreadsheet. Each action is written as a line in the sheet, starting with an action keyword followed by arguments.

A crucial step to get big testing under control with ABT is a good plan for the test modules, which takes an outline form, much like the table of contents in a book. Each test module should have a clear, unambiguous scope that is well-differentiated from the other test modules. Once the test modules have been identified, they can be developed one by one over time. The scope of each test module defines what kinds of actions to use and what kinds of checks to perform.

Of particular importance is that one avoid shaving different kinds of tests within the confines of one test module. A simple example can be taken from a recent project I reviewed. Take a set of tests on a table that presents data within a dialog. Some of the tests may involve manipulating the table, like sorting on columns or cutting and pasting rows. Other tests might address the data in the table: does it have the right values? If you were to place both such tests in the same test module, or test case, your test becomes hard to read. Moreover, it has to change when either the table navigation changes or the value calculations change. In smaller test projects, you may get away with this. But if the tests number in the thousands, or greater, you will soon find that this sort of organization makes it hard to achieve a smooth and stable automation, especially across multiple versions of the application under test.

The scope of a test module also defines how much detail you want to see in the tests, and therefore what actions you will use. In a project I looked at a few years ago, a system for bank tellers was tested. In that system the daily teller cycle would begin with obtaining an initial transaction number, which itself took several UI steps to complete. The QA manager initially insisted on having this procedure spelled out and verified step-by-step as the first part of *each and every test module* just, in his words, "to make sure". As a consequence, even a small change in the application necessitated many adjustments to the tests. Only later on was the procedure encapsulated in a high level action called, aptly enough, "get initial transaction number". As a consequence, a large share of the maintenance problems evaporated.

The automation process focuses on automating actions. Some actions may already be predefined in the tool, while higher level actions need to be recorded or developed. In the automation there are a number of things you can do to speed development and, even more effective, make the automation stable. Your developers can help by making UI elements, such as windows, controls or HTML elements, easy to identify. They can do this by assigning values to certain properties, like the "id" attribute in an HTML element, which cannot be seen by a user but *can* be seen by an automation tool. Mapping a UI then becomes something that can be performed manually and rapidly, without the need for "spy" tools.

Additionally, timing needs your attention. In large test runs on large systems, responses can vary wildly, and you certainly don't want a timeout to derail a big test run. Conversely, you don't want to slow things down due to needlessly long wait times in many places in your test. Make sure you always have something your tooling can detect and wait for. And test your actions with their own test modules, before you run the regular test modules.

**Infrastructure**

Once your tests and automation are well-designed and stable, your infrastructure basically defines how much you can do in a given amount of time. This is an area where organizations and projects can differ greatly from one another. Some companies may have substantial infrastructure in place; in others, projects may have to be built up from scratch.

As a rule of thumb, you should avoid using the machines of the test developers to also run the tests, since this may inhibit their productivity. You can either give each tester a second machine, or – probably better – run the tests on dedicated servers.

Note that although you may work with dedicated machines as servers, you may not necessarily need server hardware. Most testing is performed against user interfaces, and requires client machines with client operating systems. In particular, blade server solutions tend to be expensive, and often come with features, like load balancing for web servers, that you do not need for your playback automation.

One good way to organize execution is *virtualization* – the use of virtual machines. A virtual machine is a tester's dream. It can be set up to mimic a variety of configurations much more readily than can a physical machine. You can set up a virtual machine image with a specific operating system, a version of the application under test, and even some base data for that application already defined. Then run one or more instances of that image every time you want to test against that configuration.

One physical machine can often run multiple virtual machines in parallel, expediting your test cycle. However, when planning your infrastructure, keep in mind that automated test runs tend to place a higher load on virtual machines than does human usage. Automated runs perform operations continuously, while humans tend to be "interval-intensive". Still, it is not uncommon to have five or more virtual machines running comfortably on an inexpensive physical machine. Hardware properties that are commonly recommended for such a physical machine are: gobs of memory, multiple processors and/or multi-core processors, hardware virtualization support, and a second (physical) disk dedicated to the virtual machines it hosts.

Cloud-like infrastructures also come to mind for handling large test executions. If your company has such an infrastructure, getting a substantial part of it allocated to testing is largely a matter of making a business case for it. There are also "public cloud" vendors to consider. These can be of particular help if the need for a large test execution is only for a limited period of time. For continuous use, public clouds tend to be more costly than an in-house virtualization solution.

## *This is BIG!*

- **MMOGs** (*Massively Multiplayer Online Game*): Happy Farm is the most popular MMOG with 228 million active users, and 23 million daily users (active users logging onto the game within a 24-hour period). World of Warcraft is currently the dominant MMOG in the world with more than 60% of the subscribing player base, and with 11–12 million monthly subscribers worldwide.
- **Consumer Goods:** Walmart handles more than 1 million customer transactions every hour, which is imported into databases estimated to contain more than 2.5 petabytes of data - the equivalent of 167 times the information contained in all the books in the US Library of Congress.
- **Online Databases:** Ancestry.com claims approximately 600 TB of genealogical data with the inclusion of US Census data from 1790 to 1930.
- **Social Networks:** Facebook handles 40 billion photos from its user base. As of May 2009, Yahoo! Groups had "40 terabytes of data to index".
- **Video:** Released in 2009, the 3D animated film Monsters vs. Aliens used 100 TB of storage during development.
- **Encyclopedia:** Wikipedia's January 2010 raw data uses a 5.87 terabyte dump.
- **Finance:** In 2009, Visa's global network (known as VisaNet) processed 62 billion transactions with a total volume of $4.4 trillion. The record for VISA card transactions: 6,803 peak transactions per second (12/23/06).

### Organization

Regardless of infrastructure, the success of a big testing project may hinge on the team or teams involved. They have to develop the tests, automate them, manage their execution and follow up on results.

A big testing project has two facets that the teams must be able to handle:

- a test design and development side, with a focus on creativity, effectiveness, and efficient automation; and
- a production side, with a focus on planning, volume and timelines. This demands an industrial "get it done" attitude from the team, which is challenging in its own way.

In a scrum organization, I expect that agile teams can handle most if not all of the big testing needs. It helps to have test execution and system development in the same team, so that any issues that emerge during large tests can be addressed swiftly. However, it is important that the team also include the requisite skills and focus in regards to the managerial aspects of planning and conducting large tests.

In my experience, testing is a profession. There exists a wide variety of techniques that experienced testers can draw from to make tests lean and mean. Sadly, it is not uncommon to see organizations assume that anybody, in particular a developer, can be a tester without much training. From what I've seen, however, this mindset results in tests that are simplistic and uninteresting. This leads to shallow testing, and also to large cluttered test sets that are hard to achieve stable automation for. It behooves any organization not to underestimate a profession that has been developing for many years, with many publications and conferences to show for it.

Organizations which are faced with a need for large scale testing or large scale automation will often look at off-shoring as a way to more easily scale up and scale down based on project needs. This can be especially helpful if a large effort is needed for a relatively short period of time. However, it needs to be done with care! Big teams may be able to make big tests, but those are not always good tests. Be sure to do good planning and organization of the tests before starting, and to continuously pay attention to effective design and automation architecture.

There is no one recipe to make big testing a big success. It takes planning and careful execution of the various aspects, like test design, infrastructure and organization – a mix that can be different for each situation in which you may find yourself. ■

### 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 key-word 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.*

*Prior to joining LogiGear, Mr. Buwalda served as project director at CMG (now Logica) in the Netherlands. During his seventeen years with that firm he assisted clients in nine countries develop and deploy software testing solutions. He is an internationally recognized expert specializing in test automation, test development, and testing technology management, and speaks frequently at international conferences on concepts such as Action Based Testing, the Three Holy Grails of Test Development, Soap Opera Testing, and Testing in the Cold.*

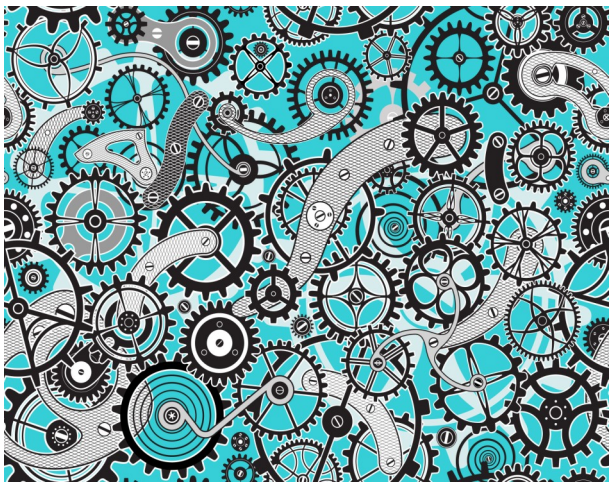
*He is coauthor of Integrated Test Design and Automation (Addison Wesley, 2001) and holds a Master of Science in Computer Science from the Free University, Amsterdam.*

Feature

# Expanding Usability Testing to Evaluate Complex Systems

*Not all scenarios have clear endings or known, correct answers. How do we evaluate the usability of systems that are too complex for our typical usability testing protocols?*

**By Ginny Redish**



When you think of usability testing, do you think about working with someone for an hour or two, watching and listening as they do a series of short, discrete scenarios where there is a clear ending or a correct answer for each scenario?

For most of us that's a typical usability test. It's the type of usability test assumed by the Common Industry Format (ISO/IEC 25062, see http://zing.ncsl.nist.gov/iusr/). It's the type being discussed for the CIF-Formative (Theofanos and Quesenbery, 2005). It's the type that the various CUE studies have focused on (see http://www.dialogdesign.dk/CUE.html).

But not all systems lend themselves to short, discrete scenarios. Not all scenarios have clear endings or known, correct answers. How do we evaluate the usability of systems that are too complex for our typical usability testing protocols?

### What do I mean by a complex system?

I am focusing here on complex information analysis: the work that domain experts do when solving open-ended, unstructured, complex problems involving extensive and recursive decision-making (Mirel, 2003, 2004; Albers 2003).

Complex information analysis takes place in many domains, including:

- Store managers evaluating inventory against future needs (Mirel, 2004).
- Corporate and government project managers allocating resources among many projects (Mirel, 2004).
- Nurses dispensing medicine (Mirel, 2004) and many other health care professionals in many situations (operating room systems, intensive care systems, neonatal care systems, patient records, and so on)
- Intelligence analysts bringing together many sources
- Emergency responders prioritizing logistics
- Train drivers (Olsson and Jansson, 2005; Olsson, Johansson, Gulliksen, and Sandblad, 2005)
- Customer service representatives who may have to interpret the customer's presentation of a problem and use multiple sources of information to help the customer
- and many others

### How do complex systems differ from those we typically encounter for usability testing?

As Mirel says (2003, 233), "complex tasks and problem solving are different in kind not just degree from well-structured tasks". These complex systems differ from the world of well-structured tasks in at least these ways:

- Data analysis and recursive decision-making are cognitively very burdensome; people have little cognitive workload available for dealing with unusable interfaces.
- Information is often incomplete. It may be unreliable. In some domains, people may have to sort deceptive from meaningful information.

- In some domains, for example, intelligence, there may be no way to know at the time of analysis if the result one gets is right or wrong.
- In many domains, for example in medicine, transportation, intelligence, and the military, time may be critical. Good decisions made too late are bad decisions. And wrong decision may have catastrophic effects. Getting it right can indeed be a matter of life or death.
- These people are typically domain experts. However, they may not be computer or systems experts. The demands of their work may make it difficult for them to put much time or effort into the learning curve of new programs or new presentation methods.
- Often, analysts and decision-makers are different people. An important question may be how data presentations and complex systems allow the people searching, gathering, and analyzing information to convey facts and interpretations to decision-makers.
- Visualizations are often a critical presentation method for complex information systems. There is a need, therefore, to study the usability of specific ways of visually representing specific types of data for specific types of users. And there is also the need to study how people develop and use visualizations in the larger context of the work they do. (Scholtz, 2006, calls this larger context the visual analytic environment.) We must evaluate components (individual visualization methods and screens) *and* the entire system (the environment in which the visualizations are used). I'll talk more later in this essay about the need for this two-level evaluation (components and entire systems).

### What else must we consider beyond ease of use?

Ease of use – what we typically focus on in usability testing – is critical but not sufficient for any product. Usefulness (utility) is as important as ease-of-use. If the product does not match the work that real people do in their real environments, it may be an easy-to-use solution to the wrong set of requirements. (This is the main point of Mirel, 2003, 2004.)

Moreover, to understand how people use systems successfully in these complex domains, we may need to include other types of evaluations beyond usability and utility (Scholtz, 2006). We may need to understand and evaluate how well the suite of tools, the presentation methods, and the entire environment support:

- Collaboration among users (and between users and others, such as decision makers, who may not themselves use the system)
- Creativity and innovation (Fischer, 2005)
- Interaction (of the user with the same system over time or with a variety of systems that should – but may not – interconnect)
- Iteration (the same user returning to the system, wanting to retrieve previous analyses or records, and so on)
- Reduction of human error (For a review of various methods proposed to study this issue, see Shorrock and Kirwan, 2002. For an example of applying predictive human error analysis [PHEA], see Parush, et al., 2004.)
- Situation awareness (Endsley, 2000; Endsley, Bolté, and Jones, 2003)

### Why have we not done more about new usability testing techniques for these complex systems?

Within the usability community, the focus in considering these complex, open-ended systems has, quite correctly, been in pre-design studies – in understanding the domain experts' work. We want designers and developers to get it right beforehand, not to try to fix it through evaluation later.

Pre-design usability studies are absolutely necessary. However, they are not enough. All design and development projects require evaluation as they move from concept to prototype to functioning system. We still need formative evaluation (usability testing) techniques for complex information systems as they are being designed and developed.

### What does this mean for us as usability specialists?

Here are just some of the points we must consider as we expand our usability testing techniques for complex information systems for domain experts:

### Collaborating with the domain experts

Most of us are usability or design or communication experts. We are not experts in the domains I am talking about in this essay. And becoming expert in these domains is not a trivial undertaking. This makes it very difficult to apply user-free formative evaluation techniques in which the usability specialist serves as surrogate user, such as cognitive walkthroughs (Polson, Lewis, Rieman, and Wharton, 1992) and persona-based / task-based / heuristic-based evaluations (Chisnell, Redish, and Lee, 2006).

In all evaluations, working with the client to understand the potential users and their scenarios is very important. In the situations we are discussing here, it is critical. The domain experts must be partners in the evaluation, just as they must be partners throughout the planning, design, and development of the systems.

### Collaborating with other specialists

If we, as usability specialists, concentrate on issues of effectiveness, efficiency, and satisfaction, we may have to work with specialists who focus on utility, collaboration, creativity and innovation, interaction, iteration, and situation awareness to get the overall evaluation that is necessary to ensure a successful system. Evaluations like these should not be done in assembly line fashion being passed from one specialist to another but in team work, as a collaborative endeavor.

Furthermore, in many of these domains, security and privacy are major issues. In domains like intelligence and medicine, issues arise about dealing with real data about real events and real people or trying to set up entire data sets with surrogate data.

### Getting the right scenarios

In all usability tests, we want realistic tasks. For these complex domains, how will we as non-experts even be able to define good tasks unless we work with the domain experts? And the tasks must be complex enough to represent the realities of the world in which the systems will actually be used. How do we get the right level of complexity? How do we set up usability testing with the time and environment that is realistic? As Caroline Jarrett says, for these complex domains, teams must "go into the field and trap cases 'in the wild' to use as tasks for usability evaluations" (email to author, 1/16/07).

### Getting the right users

We know how critical it is that our usability test participants represent the people who will use the system. We usually worry about getting a false positive result – that the product does fine in usability testing, but when it gets to the users, they have lots of problems with it. A false negative result – that the product shows lots of problems in usability testing, but the real users would not have those problems – is also worrisome. False negatives may be more likely in the

systems we are considering here – if we do not have the domain experts as usability test participants.

Relevant issues, therefore, include:

- How do we get the time of the domain experts to participate in usability testing?
- How much incentive (money or other) is necessary to engage their participation?
- If surrogates must be used, what are the essential characteristics of the targeted users that we must match?

### Understanding how difficult it may be to set goals and tasks

For a usability test of any of these complex systems, we are likely to be able to specify the users and the context of use. The goals, however, will usually be at a higher level than typical usability testing goals, and they may be much harder to specify.

Furthermore, these initial, high-level goals may be vague, such as: "What in this patient's records will help me understand how to interpret this patient's current complaint and relate that to the patient's overall heath?" or "What are we overstocked on and would putting that on sale be good for our bottom line?" or "Is there a trend in this data that I should make my boss aware of?"

These goals (and especially the subgoals to achieve the larger goal) are likely to change as our domain expert moves through the data. Also, our domain expert may be trying out "what if" scenarios using the data to explore aspects of or possible solutions to the larger goal.

In the context of the possibly vague and almost certainly shifting goals of a complex information analysis, it may be very difficult to define *a priori* what constitutes effectiveness or efficiency in a

given scenario. In almost all information gathering and analysis tasks, people satisfice. They stop at a point where they are satisfied enough with what they have achieved.

### Accepting experts' judgment of completion and effectiveness

If we do not ourselves know the entire data set that is being used for the usability test, we may well not know the answer to a given task. We must rely on the usability test participants' judgments that they have arrived at a reasonable solution. In still other domains, again such as intelligence or medicine, the rightness of an answer can only become known over time. What do we set as a measure of success for our usability test?

### Doing usability testing of both components and entire systems

Systems for complex problem solving often include many pieces (components, tools). Usability testing at the component level may be possible and very useful for some situations. And our typical usability test protocols will often work for testing specific components.

However, we must also test the suite of components together at some point because the only true measure is the user's success at solving the problem, however large that problem may be. And typical usability testing is too short, too "small task"-based, and not context-rich enough to handle the long, complex, and differing scenarios that typify the work situations that these complex information systems must satisfy.

### What might we do

No single methodology or measure is going to work for usability testing of all these systems for domain experts doing open-ended problem solving. As Mirel reminds us (2003, 250): "To be analytically useful, interactive data visualizations have to be designed to allow users to employ and see the results of the analytical methods relevant to the lines of reasoning in their particular area of specialty for a given type of problem. These lines of reasoning are not generic. They are social and contextual." And Mirel's point is valid for all systems, not just those that use interactive data visualizations.

(Scholtz, Morse, and Potts Steves [2006] began to develop a list of dimensions and factors along which these complex information systems vary. Redish and Scholtz [2007] expanded that list.)

For usability testing of complex information systems, some of the techniques we might consider include:

- Conducting usability studies outside of the laboratory, for example at conferences where designers, developers, and domain experts meet
- Multiple evaluators to observe different team members in collaborative work
- Simulations (with consideration of how well the simulation captures enough richness and complexity of the real work)
- Situation awareness assessments (for domains where that is appropriate)
- Think aloud (especially cued retrospective where concurrent think aloud would pose too much additional cognitive workload) Redish and Scholtz (2007) include an extensive discussion of the research on various types of think aloud and implications for usability tests of these complex systems.
- Unattended data capture for portions of a long-term evaluation, used along with observations and interviews.

### What has been tried?

Here are four very brief case studies. They are all from research projects, not from usability testing of commercial systems, but they can give us ideas for expanding our usability testing techniques.

### Testing with simulated situations within a typical usability testing time frame

Patterson (1999) reports on a study to find out whether expert intelligence analysts, working on a topic that was not their primary specialty, would find, select, and use the best sources available to answer a given question when they had a large data set of documents and a short time frame. In this case, the researchers knew which were the best documents and what the analysts should report. What they learned was how different analysts searched, what data they kept, how much time they were willing to spend, and how much they relied on their own knowledge compared to using the data in the documents.

### Taking advantage of conferences and contests

Contests have been used in several domains as a way of focusing attention and evaluation on components for moving a field forward. For example, the Message Understanding Conference (MUC) was started in 1987 to do qualitative evaluation of the state of the art in message understanding, and the InfoVis Contest was begun in 2003 to create an Information Visualization Repository of resources

to improve the evaluation of information visualization techniques and systems (Thomas and Cook [Eds.], 2005, 152).

Based on the success of contests in these other domains, the Visual Analytics Science and Technology (VAST) conference, held in the Fall of 2006, included a contest as a way of bringing developers together with domain experts. Commercial organizations and university teams that are researching and developing visual analytic tools showed how well (or poorly) their systems worked for a problem that the contest organizers set.

The teams whose software did best in a first round where the developers acted as users then got to see how their systems worked for actual domain experts. A domain expert was assigned to each team to use the system to work on a second problem that the organizers set. So this was a sort of usability test at a conference with the added flavor of different development teams competing to be the most effective, efficient, and satisfying system to complete the assigned task. (Grinstein, et al., 2006; www.cs.umd.edu/hcil/VASTcontest06).

### A week-long formative evaluation in a real environment

Scholtz, Morse, and Potts Steves (2006) report on a study that included week-long evaluations. Volunteer intelligence analysts participated for two weeks. In the first week, they were trained on the new system and then spent several days on a practice task. After they finished the practice task, they were tested to ensure that they could use the basics of the new system.

They were then given a week to research a particular question and generate a report on it – typical of the work that they do. Analysts were debriefed daily, in person on the first day and through an online form during the rest of the week. Further data was collected through the Glass Box (Cowley, Nowell, and Scholtz, 2005).

The Glass Box is software that captures keystrokes, links followed, and search terms used. It can gather a continuous recording of the computer displays along with audio from participants. All entries are time stamped. Users can make annotations at any time.

The users here were specifically asked to explain what they were doing any time they were away from the system for 15 minutes or more – in part to capture times when they were gathering data from people or paper or analyzing information offline. (Scholtz [in Redish and Scholtz, 2007] notes that data from the Glass Box can be difficult to analyze as it is in complex relational databases that require scripts to sort out.)

An evaluation like this is more of an instrumented pilot release than a typical usability test, but it did allow the evaluation team to see the entire analytic process, see what features of the software were used and when, see what documents the analysts read, what information they took from which documents, etc.

### Formative evaluation in a partnership with domain experts

In the Scandinavian tradition of participatory design (Schuler and Namioka, [Eds.],1993; Greenbaum and Kyng, 1991), domain experts are part of the design and development team throughout the process. Olsson, Johansson, Gullikssen, and Sandblad (2005) describe several participatory projects with domain experts. One of those projects is TRAIN (Traffic Safety and Information Environment for Train Drivers).

In the TRAIN project, the Uppsala University researchers spent three years doing an ethnographic study of train drivers at work. They then involved six train drivers, with a spread of experience, gender, and company they worked for, in iterative participatory exploration and design of a new interface for the engine cab of Swedish trains. System engineers and two HCI researchers worked with the train drivers. (The design phase of the project is discussed in Olsson and Jansson, 2005.)

At the end of the 2005 paper, Olsson and Jansson said that their next step was evaluating the prototype in a simulator, comparing it to the currently existing system and measuring performance and situation awareness. In early 2007, Jansson informed us (email to the author, 2/1/07) that their first evaluations showed that it is difficult to get a complex enough scenario in a simulated environment to capture the expertise of the domain experts, and, also, that it is difficult to operationalize situation awareness in the train domain. They are working on developing a simulation that is complex enough and that will allow train drivers to act close enough to the way they do on their jobs.

### Conclusion

Many of us have expanded our repertoire of usability testing techniques beyond the controlled laboratory setting. Many of us would agree with Joe Dumas (Dumas,

2003) that usability testing today is not a single technique but a spectrum of related techniques. However, the much broader considerations needed to assure success of complex information systems for domain experts doing open-ended, recursive analysis may require measures and methods beyond even the wider spectrum of usability testing techniques that we have so far developed. These are critical systems. We should be much more involved with them than we have been. ■

**Acknowledgements**

I thank Caroline Jarrett, Avi Parush, Jean Scholtz, and Whitney Quesenbery for comments on earlier drafts of this essay.

**References:**

Albers, M. J., 2003, Complex problem solving and content analysis, in M. J. Albers and B. Mazur (Eds.), *Content and Complexity: Information Design in Technical Communication*, Mahwah, NJ: Lawrence Erlbaum Associates, 263-284.

Chisnell, D., Redish, J. C., and Lee, A., 2006, New heuristics for understanding older adults as web users, *Technical Communication*, 53 (1), February, 39-59.

Cowley, P., Nowell, L., and Scholtz, J., 2005, Glass Box: An instrumented infrastructure for supporting human interaction with information. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 9 - Volume 09*.

Dumas, J. S., 2003, User-based evaluations. In J. Jacko and A. Sears (Eds.),
*The Human-Computer Interaction Handbook*, Mahwah, NJ: Lawrence Erlbaum Associates, Inc., 1093-1117.

Endsley, M. R., 2000, Theoretical underpinnings of situation awareness: A critical review, in M. R. Endsley and D. J. Garland (Eds.), *Situation Awareness Analysis and Measurement*, Mahwah, NJ: Lawrence Erlbaum Associates.

Endsley, M. R., Bolté, B., and Jones, D. G., 2003, *Designing for Situation Awareness – An Approach to User-Centered Design*, Boca Raton: CRC / Taylor & Francis.

Fischer, G., 2005, Creativity and distributed intelligence, *Report of the NSF Workshop on Creativity Support Tools*, 71-73, downloaded from http://www.cs.umd.edu/hcil/CST.

Greenbaum, J. and Kyng. M., 1991, *Design at Work: Cooperative Design of Computer Systems*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Grinstein, G., O'Connell, T., Laskowski, S., Plaisant, C., Scholtz, J., and Whiting, M., 2006, VAST 2006 Contest – A Tale of Alderwood, *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology*, 215-216.

Mirel, B., 2004, *Interaction Design for Complex Problem Solving – Developing Useful and Usable Software*, San Francisco: Morgan Kaufmann.

Mirel, B., 2003, Dynamic usability: Designing usefulness into systems for complex tasks, in M. J. Albers and B. Mazur (Eds.), *Content and Complexity: Information Design in Technical Communication*, Mahwah, NJ: Lawrence Erlbaum Associates, 233-262.

Olsson, E. and Jansson, A., 2005, Participatory design with train drivers – a process analysis, *Interacting with Computers*, 17, 147-166.

Olsson, E., Johansson, N., Gulliksen, J., and Sandblad, B., 2005, *A Participatory Process Supporting Design of Future Work*, paper from the Department of Information Technology, Human Computer Interaction, Uppsala University, Sweden, retrieved from http://www.it.uu.se/research/publications/reports/2005-018/2005-018-nc.pdf

Parush, A., Erev-Yehene, V., Straoucher, Z., Rugachev, M., Kedmi, E., and Markovitch, R. (2004) *The safety implications of command and control tasks: A comparative study*, Research Center for Work Safety and Human Engineering, Israel Institute of Technology, HEIS-04 (available in Hebrew from the first author at Avi_Parush@carleton.ca).

Patterson, E. S., 1999, A simulation study of computer-supported inferential analysis under data overload, *Proceedings of the Human Factors and Ergonomic Society, 43rd Annual Meeting*, 363-367.

Polson, P. G., Lewis, C., Rieman, J., and Wharton, C., 1992, Cognitive walkthroughs: A method for theory-based evaluation of user interfaces, *International Journal of Man-Machine Studies*, 36, 741-773.

Redish, J. C. and Scholtz, J., 2007, Evaluating complex information systems for domain experts, paper presented at a symposium on *HCI and Information Design to Communicate Complex Information*, Memphis, TN: University of Memphis, February. (Paper available by request to the first author at ginny@redish.net.)

Scholtz, J., Morse, E., and Potts Steves, M., 2006, Evaluation metrics and methodologies for user-centered evaluation of intelligent systems, *Interacting with Computers*, 18, 1186-1214.

Schuler, D. and Namioka, A., (Eds.), 1993, *Participatory Design: Principles and Practices*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Shorrock, S. T. and Kirwan, B. (2002) Development and application of a human error identification tool for air traffic control, Applied Ergonomics, 33, 319-336.

Theofanos, M. and Quesenbery, W., 2005, Towards the design of effective formative test reports, *Journal of Usability Studies* 1 (1), November, 28-46.

Thomas, J. J. and Cook, K. A. (Eds.), 2005, *Illuminating the Path: The Research and Development Agenda for Visual Analytics*, IEEE Press.

Reprinted with permission from the *Journal of Usability Studies*, © UPA, *Journal of Usability Studies*, May 2007, Volume 2, Issue 3, pages 102 - 111.

## About Ginny

*Janice (Ginny) Redish has been a passionate evangelist for usability and clear language for many years. She is an internationally-renowned consultant and speaker, widely recognized for her pioneering work in all aspects of user experience (UX).*

*Ginny is co-author of two of the classic books on usability and author of numerous papers and book chapters on usability, writing for the web, plain language, and other topics. Reviewers have raved about her most recent book, Letting Go of the Words – Writing Web Content that Works. www.redish.net; Twitter: @GinnyRedish*

# The Seven Most Stunning Website Failures of 2011

*By Apica - Load test & Web Performance monitoring - www.apicasystem.com*

Apica, a leading load testing and performance -monitoring provider for cloud and mobile applications, recently offered its list of the "Seven Most Stunning Website Failures of 2011."

"This year's list proves that ongoing website failures and performance problems are costing companies lost revenues and damaged reputations. As you can see, even the biggest in the business are suffering. When a website goes down, an online business has effectively shut its doors and is left wondering 'Will that visitor ever come back?,'" said Sven Hammar, website performance optimization expert and CEO for Apica. "Website failures and performance problems can be minimized greatly by simply, conducting on-going testing and monitoring to avoid being the next big failure story."

### 1. Tickets for London 2012 Olympics available for purchase


In June, the Olympic committee announced 2.3 million tickets were available for purchase for the 2012 London Olympics. Excited fans rushed the site to find "Sorry, we cannot process your request at this time" messaging because the website could not handle the rush of visitors. London has gone to extraordinary lengths to win and host the Olympics, only to have its website crash when it opened its doors to the public.

### 2. Target.com launches Missoni Collection


The much anticipated launch of the Missoni Collection at Target in September was a huge failure when the Target.com site crashed due to high traffic, rendering the entire website inoperable for hours. Target experienced a repeat crash performance six weeks later when another rush of visitors hit the site. Target has stunning marketing, but surely having their website fail during such a high profile event isn't part of their plan.

### 3. Apple launches iPhone 4S


In early October, iPhone 4S became available for purchase at the Apple Store. Huge spikes in website traffic throughout the day either downed the site or slowed it considerably, effectively halting tens of thousands of new phone purchases. While everyone was hoping for the iPhone 5, surely Apple had enough confidence in the 4S to prepare for enormous interest.

**4. Bank of America announces monthly fee for debit cards**


Bank of America's website slowed considerably for five days in October when it was hit hard by customers flocking to the site after the banking giant announced it would be charging a $5 monthly fee for account holders who make purchases with their bank debit cards. While Bank of America could not have imagined the worldwide "Occupy" movement stemming from this announcement, the company had to have spent enormous time considering the public response — but not the potential traffic to their website.

**5. New York City's government Website for Hurricane Irene**

The New York City website crashed amid overwhelming visitor volume encouraged by Mayor Michael Bloomberg who strongly urged the public to use the website for information on Hurricane Irene in August. More than any other city in the world, you would think that New York City would be most prepared for extraordinary events.

**6. Disney Store launches limited edition princess dolls**


Disney.com crashed repeatedly due to high visitor volumes when it launched the limited edition of its princess dolls collection in October. Even the world's most amazing merchandiser can get caught off guard with a potential hit on their hands. Simple proactive testing for enormous peak loads for an expected 'big seller' and this situation would be easily avoided.

**7. H&M launches new Versace Collection**


Shoppers looking to purchase the new Versace Collection from H&M's website in November were greeted with "We're sorry, we are experiencing large number of visitors at the moment, please try again later" messaging. "Try again later" is the last thing this retailer's avid fans want to read. Most of its designer lines are smash hits. ■

## Spotlight Interview

# A Big and Complex Interview

***Michael Hackett** talks to three Salesforce.com quality engineering directors about working for one of the world's most exciting and fast-growing companies.*

*For this interview, we talked to Greg Wester, Senior Member Technical Staff, Craig Jennings, Senior Director, Quality Engineering and Ritu Ganguly, QE Director at Salesforce.*

*Salesforce.com is a cloud-based enterprise software company specializing in software as a service (Saas). Best known for its Customer Relationship Management (CRM) product, it was ranked number 27 in Fortune's 100 Best Companies to Work For in 2012.*

**What is big or complex about your system (users, physical size, data, load, distribution, safety, regulation, security, other)?**
This should give you an idea. Salesforce processes 700 million highly complex business transactions per day for nearly 3 million active users, whose raw processing needs are growing at a 50% compounded annual rate. We expect to soon exceed 1 billion transactions a day across global 6 data centers housing 20 computing clusters, which we call "pods". We have a multitenancy architecture where each customer's data lives with a group of other customers in one of these pods. Within a pod we have a horizontally scaled application tier on x86 commodity hardware that, among other things, also hosts a distributed cache. We have a home grown message queueing system that allows asynchronous processing to be scheduled in either the database tier or the application tier.


**Do you remember any remarkable event that changed your mind about how big or complex your system is?**
This company has a strong culture of "putting our money where our mouth is", so it should be no surprise that we use our own product to run our business. It's also well-known that Salesforce employees collaborate, share, and align on our corporate social networking product, *Chatter*. When *Chatter* was still under development, our founder, Marc Benioff, encouraged every employee to share their vision and goals document on their Salesforce profile on Chatter. As a result, one of our non-customer facing pods showed a brief performance decrease while we added physical storage to the file servers. This affirmed that monitoring and management tools are often as important as the product software itself in achieving high uptime. You have to watch what's happening, you have to respond quickly, and you have to learn from what's happened. Our early movement towards being an open social enterprise exceeded estimations. However, we were prepared by our DNA of using our own product to run our own business.

**Do you document testing as you have in the past, or has documentation become leaner even with a big or complex system?**
At Salesforce, we think automated test cases describe how a feature works far better and more efficiently than a design document. We're an Agile shop, so our design documentation isn't voluminous. However, the aggregate of tests that have passed and failed, are a more current, accurate, detailed, and up-to-date description of our product than any written test documents. We have made it more efficient based on our customer's needs. We have reviewed traditional test plans/strategies and kept what is needed but our philosophy is lean: less documentation and more testing. That's not to say we don't do test planning. Our Quality Engineers must first think about the feature at a high level. We built a tool that encourages thinking about the feature at a high level. You must understand the customer's use cases, list out all your assumptions, and plan out a testing strategy is. After this though, we get the engineer right into their coding environment and keep them there. They'll stub out their test cases and write the intent of the test case and expected result in Javadoc. When we check those tests in, we have a tool that parses the Javadoc, and sticks the test case name, description, expected result into our test case repository automatically. It's quite intelligent and keeps the engineer productive since they don't have to switch contexts at all.

**What type SDLC do you follow? Have you found limitations in SDLC due to the size of the system you support?**
Salesforce has its own flavor of Agile called the Adaptive Delivery Methodology, or ADM. It's pretty much textbook Scrum with a few twists. Product Owners prioritize features from a backlog based on customer interest, and business opportunity. Teams of four to a dozen engineers in development, quality, performance, user experience, and documentation meet in a daily stand up meeting and collaborate to deliver "potentially releasable" features each iteration, which can be chosen by the

individual team. Most are on two-week iterations, but some are on week-long sprints. Our customer base requires that we introduce features with ample notice and staging beforehand on sandbox environments. We are very careful about what goes into a patch release, because a fix for one customer can turn out to be a bug to another.

**What is the biggest problem you face in delivering your system to users?**
Our platform is basically an ecosystem that is built and managed by us, but controlled by the customers. Our sales are increasing at over 38% year over year. As a result of this success, the performance tuning tweaks we've verified and deployed to our system today may be suboptimal a year from now, even if we made no major code changes. Scale is key. We know we have some of the best Technical Operation and R&D teams in the world. Their coordinated success is ultimately the foundation of our business model.

**How has your testing strategy changed as your system got bigger or more complex?**


Definitely, we have had to look at our customer's complex implementation needs, complex business processes and customizations and ensured we represent real customer scenarios in our testing. As we grew at an enormous rate, we learned how support cases that escalate to R&D drag the velocity of feature work. They mire teams in bug fixing. Too many bug fixes in patch releases also introduces risk to the product. The goal of our testing strategy is to minimize the amount of time supporting the feature after it's released. In other words, our aim must be to find all of the impactful bugs, corner cases, and quirks. We leverage tests written by customers in our Apex Code language to verify their use cases before each major release. Since some bugs are inevitable even with a very thorough process, we then put effective monitoring and management systems in place so that we can react to issues immediately when they arise.

**Did you change the experience or job requirements for test engineers as a result of a bigger or more complex system?**
Salesforce's customers have expectations that our system will have minimal downtime each year, and no unscheduled downtime. We are delivering a service at a scale where every test must be automated, and most features have more SLOCs of test code than application code. We hire only software engineers into Quality Engineering who can perform white and black box testing. Not every development engineer has the instinct to be a test engineer, and vice versa. Quality Engineering requires solid programming skills, a laser focus on customer service, a knack for risk management, and an eye for hidden or low frequency/high impact bugs. We have also expanded our performance testing team.

**Have you changed your reliance on test automation due to size or complexity?**
We rely on it in increasing amounts. This is the only way we can continue scaling. We think test automation has a maturity model by which you can measure the commitment to quality within an organization:

- Level 1 is unit test coverage, representing a certification by the developer that individual implementations of software classes function in a particular way.
- Level 2 includes functional testing of a module of software classes.
- Level 3 is end to end testing of every module in a particular application while it is running on a single host or node.
- Level 4 is testing the application under load for an extended period with all of its supporting subsystems including database, cache, message queues, etc.
- Level 5 has the same parameters as Level 4, with the added requirement that every piece of hardware, every operating system library, and every configuration is as it appears in a production environment with customer data.

When we start seeing diminishing returns from one level, we move to the next.

**What percentage of your tests are automated?**
Over 90%. Our goal is that no teams run manual tests, unless you're counting on exploratory testing (which every team does). That last 10% is amazingly difficult. We're forced to test manually when the tools for automation are in their infancy, such as on Mobile platforms. We've made impressive leaps forward in these areas, but there's still work to do. Talk to us next year. We'll have solved some of those problems and be closer to 100%

**What do you see in the future for testing big or complex systems?**
Mainstream tools like JUnit were designed for unit testing a single class but have evolved to accommodate complex functional testing scenarios. Unit testing on a simple piece of code has a binary outcome: pass or fail. Functional testing on a live distributed system with components designed on loose service level agreements to accommodate graceful degradation and failure of neighbors is different. The tools for this are in their infancy and require engineers as creative and talented as the ones who designed the system to write frameworks for testing it. This presents an opportunity for a thought leader to emerge with an industry standard for distributed software testing. We're proud of our accomplishments in this area, and are aiming for that goal. ■

# Glossary: Big Testing

**Ultra-large-scale system**
An ultra-large-scale systems (ULSS) is one which has the characteristics of:

- operationally independent sub-systems;
- managerially independent components and sub-systems;
- evolutionary development;
- emergent behavior; and
- geographic distribution.

In addition to these attributes, a Northrop report argues that a ULSS:

- features decentralized data, development, evolution and operational control;
- addresses inherently conflicting, unknowable, and diverse requirements;
- evolves continuously while operating, with different capabilities being deployed and removed as warranted;
- contains heterogeneous, inconsistent and changing elements; and
- treats failure as the norm, rather than the exception, with it being extremely unlikely that all components are functioning at any one time.

Examples of ultra-large-scale systems include: the US healthcare system, US energy smart grids and global financial markets.

**Real-time system**
A system is said to be *real-time* if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed. Real-time systems, as well as their deadlines, are classified by the consequence of missing a deadline:

- **Hard**: Missing a deadline is a total system failure.
- **Firm**: Infrequent deadline misses are tolerable but may degrade the system's quality of service.
- **Soft**: The usefulness of a result degrades after its deadline, thereby degrading the system's quality of service.

Soft real-time systems are typically used where there is some issue of concurrent access and a need to keep a number of connected systems up to date with changing situations. A good example of this is software that maintains and updates the flight plans of commercial airliners. Such plans must be kept reasonably current, but can operate with a latency of seconds.

**Massively multiplayer online game**
A massively multiplayer online game (also called MMO or MMOG) is a multiplayer video game which is capable of supporting hundreds or thousands of players concurrently.

**High volume and high capacity**
The primary goal of volume and capacity testing is to ensure that IT capacity meets current and future business requirements in a cost-effective manner.

**High availability system**
High availability is a system design approach and associated service implementation that ensures a prearranged level of operational performance will be met during a contractual measurement period.

Unavailability is most often called "downtime" and its avoidance requires redundancy, fail-over, or load balancing.

People commonly refer to 24/7, but a high availability system is characterized by its specified availability level. A typical spec, what's popularly known as “four nines availability”, is 99.99% uptime, allowing for 52.56 minutes of downtime per year. Systems such as these are used worldwide in the most important applications such as medical informatics, nuclear power, financial institutions, aeronautics, airlines, news reporting, e-commerce, online transaction processing and persistent online games.

**Life-critical system or Safety-critical system**
A life-critical system or safety-critical system is a system whose failure or malfunction may result in one or more of the following:

- death or serious injury to people
- loss or severe damage to equipment
- environmental harm

Risks of this sort are usually managed with the methods and tools of safety engineering. A life-critical system is designed to lose less than one life per billion ($10^9$) hours of operation. Typical design methods include probabilistic risk assessment, a method that combines failure mode and effects analysis (FMEA) with fault tree analysis. Safety -critical systems are increasingly computer-based.

**Innovation**
The act of introducing something new, such as a new method or device.

**Predictability**
To state, tell about, or make known in advance, especially on the basis of special knowledge.

Innovation and predictability are often seen at odds since innovation and change will reduce predictability and increase risk. The amount of added risk will increase significantly due to their size, complexity and emergent nature.

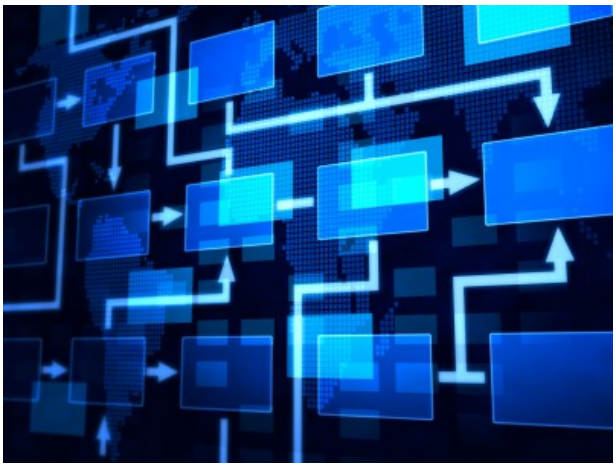
*Sources: Webster’s Dictionary, Wikipedia*

Feature

# How to Write a Great Software Test Plan

*With complex software systems, you can never test all of the functionality in all of the conditions that your customers will see. Start with this as a fact:* ***You will never test enough!***

**By Robert Japenga, MicroTools**



Step 2 in getting started is to read and re-read *The Art of Software Testing* by Glenford Myers. This classic will set the stage for understanding some of the strategies and methodologies for testing software. He provides a good definition of a number of terms we will be using in this article. Myers makes a simple statement that most of us miss when we test:

> *[Software] Testing is the process of executing a program with the intent of finding errors.*

There is big difference between testing to show that something works versus testing to show that something fails.

Step 3 is to follow these guidelines and you will be home free.

### Why Write a Software Test Plan?

A better question is why plan anything? You plan those things that you know are more complicated than what you can do by the seat of your pants. You plan things for which order and completeness are important. Even the simplest software system is so complex and is so prone to failure, that planning to test is just as essential as planning your design.

The second major reason is that, as an industry, we are doing a lousy job at releasing quality products. Current industry standards are that tested and released software averages more than ten significant bugs per 1000 lines of code. How many hundreds of thousand of lines was that last project? Thinking ahead and planning your testing is one way to cut that down.

All of these are good starting reasons for having a software test plan.

### What is the difference between a Test Procedure and a Test Plan?

Most good software development models call for both a Test Plan and a Test Procedure. Although in principle this sounds good, test procedures are extremely costly to develop, costly to maintain and don't catch enough of the bugs in a short enough time.

Simply put - a test plan tells you what you are going to test while a test procedure tells you how you are going to test it. And a quality test plan, as outlined in this article, will cover a large percentage of systems being delivered today without the need to develop a test procedure.

### Software Test Plan Part 1

This part falls under the more conventional definition of software test plans and includes:

1. **Definition and Objectives of the Phases of Testing - This should answer the questions:**

- Will there be Module Testing? If so, what are the objectives?
- Will there be Integration Testing? If so, what are the objectives?
- Will there be Systems/Acceptance Testing? If so, what are the objectives?
- Will there be Beta Testing? If so, what are the objectives?

2. **Schedules - This should answer the questions:**

- When will Part 2 of the Test Plan be written for each Phase?
- When will each of the Phases be scheduled?
- What are the dependencies of each Phase?

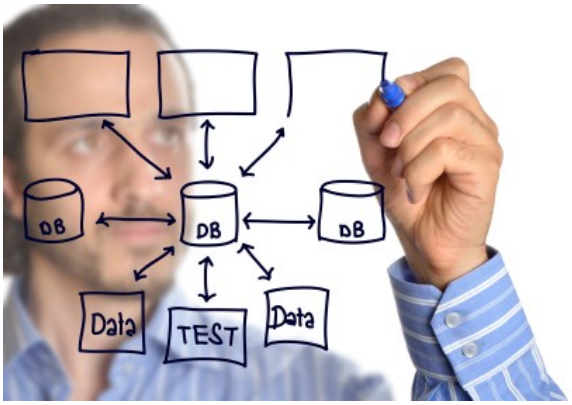
**3. Responsibilities:** For each phase, the people who will design, write, execute, and verify test cases as well as the people who correct the problems should be identified.

**4. Target Equipment Required:** All of the target equipment required for testing should be identified. This is a much neglected part of the process. If two prototypes are required, they need to be planned very early. Will you be testing on one prototype and one pre-production piece of hardware? On how many shifts will testing take place?

**5. Test Equipment Required:** All of the test equipment required for testing should be identified. Any off the shelf test equipment should be identified as well as any test equipment that will need to be created. The plan should identify what test equipment is already owned and what will need be procured either by purchasing or renting.

### Software Test Plan Part 2

This is the part that we have found can replace the Test Procedure. The goal in this phase is to define exactly *what* should to be tested. And what needs to be tested but the requirements from the great SRS you created (as a result of reading our "How to Write a Great SRS").




Here are the guidelines we follow in creating this important phase:

- Create a document which is structured identically to the SRS
- Take every requirement from the SRS and create one or more test cases from that document.
- Provide a check box beside each test case. This allows the Test Plan to easily become a Test Report.
- Create stress testing scenarios that step outside of the SRS and look at the system as a whole. For example, if the system has TCP/IP Ethernet connection on it, perform certain tests with large amounts of data being sent to the system. Or, if the system has a keyboard as input, perform testing with the keyboard being pounded with both valid and invalid data.
- Provide off-design test scenarios outside of the SRS that test the system in conditions for which the SRS provides no definition. Showing that the software does what it's supposed to do is only half the battle. The other half is determining if the software is doing what it is not supposed to do.

### Test Case Generation

The following are some general guidelines for creating test cases:

- Every test case should define expected results as a result of the defined inputs.
- Every requirement should be tested at and beyond its boundary conditions (for example: high and low limits) as well as mid points.
- Re-read Myers on Test Case generation. Then re-check your test cases.

Mix black box and white (or glass) box testing. This means that some test cases are written without knowing what is inside the box (black box testing). Other testing should include testing based on knowing what is inside. For example, if you know that an algorithm for generating 10,000 different outputs is done with a formula, you will not need to test all 10,000 outputs. However if you know that it is done with a table with some break points in the table where different algorithms are used at each break point, you will want to include this in your testing. Also, white box knowledge can reduce the number of actual tests necessary through equivalence partitioning. (Don't know what equivalence partitioning is? Go To Step 3 and repeat!)

### Who should execute the Test Plan?

We have found that it is essential to have a different person or team execute the Test Plan from those who created the software. Find individuals who like to break things! Develop a creative tension between development teams who test each other's software. ■

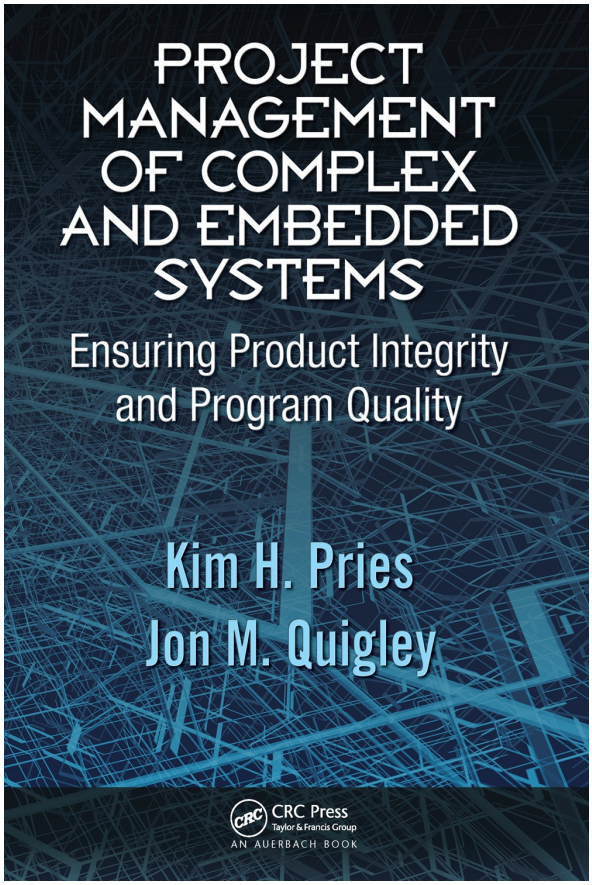
### About Bob

*Bob Japenga has been designing embedded systems since 1973. In 1988 he started MicroTools with his best friend. MicroTools specializes in creating a wide variety of real time embedded systems. With a combined embedded systems experience base of over 150 years, they love to tackle impossible problems together. Bob has been awarded 11 patents in a variety of areas related to embedded systems and motion control. Bob and his wife Barbara live in Simsbury Connecticut. They have 3 daughters and 8 grandchildren. In their spare time, they are co-directors of the Center for Renewal Retreat Ministry (http://retreats.cpcbarn.org). You can reach Bob at rjapenga@microtoolsinc.com.*

# Book Review

## Project Management of Complex and Embedded Systems

Marcin Zręda




I am not a big fan of concepts which moves industry standards to IT. I am rather an Agile and Scrum guy. Managing multiple projects at once and trying to set a high quality standard is a challenge and this book shows how industrial language can be translated into software development. I do not think that it is an IT project Bible but it is some kind of handbook for multiple project management purposes. Not easy and not obvious but worth reading to imagine some structural and complicated issues. The second part of the title is "Ensuring Product Integrity and Program Quality" – great catchword, let's see if there is an answer.

Authors Jon M. Quigley and Kim H. Pries are well known project management authorities. Kim is an author of book regarding Six Sigma for New Millennium, Jon is a project management consultant with many years experience in software and hardware development. The book written by practitioners is compact, concise and based on the problems and solve them. Quickly get answers for your questions. There are many charts, diagrams and schemas. "War stories" that end every chapter give us concrete solutions to real life issues.

I will explain a few chapters, which were taken especially in my memory. Every project manager will find something valuable.

### 1.1.7 Project Manager's Role

This is a great, ten page long chapter dealing with the described role components like customer focus, brainstorming, team creation and conflicts resolving. It shows different organization structures: functional and matrix. It also shows the multiplicity of channels of communication and exchange valuable ideas in the form of checklist.

### 3 Concept

What is more important, if not the idea? What is more important than the well-described and validated idea? This section thoroughly describes each component of the complex process of creating the concept and refining the requirements. I cannot say how easy it is to analyze, and respond to change - these issues are determined with a sort of Agile methodologies, but describes things at a higher level, help to better understand this most important stage of production.

### 7 Release to Production

This is my favorite part, release management is an issue that recently had to do in life. I know how difficult it is to truly organize this stage of the project, which has a crucial impact on the quality of the final product. Described points are taken from the industry, but very well suited for the production of software, you will find here: Trial Productions Runs, Pilot Runs, Method Builds, Production Release Risk and Costs. ■

*Project Management of Complex and Embedded Systems is available at Amazon.com.*

# Demographics of the Respondents

*__Michael Hackett__ looks at the professional and demographic details of our survey respondents.*

*In this installment of the 2010-2011 Global Testing Survey, we analyze the demographics of the more than 100 respondents from 14 countries. The next and final installment will analyze the "For Managers" section of the survey.*

**1. Job Title - for people who test**

| | Response Percent | Response Count |
|---|---|---|
| QA | 12.6% | 13 |
| QC | 1.9% | 2 |
| QE | 3.9% | 4 |
| Tester | 4.9% | 5 |
| Test Engineer | 8.7% | 9 |
| Quality Analyst | 4.9% | 5 |
| V&V Engineer | 1% | 1 |
| Analyst | 1% | 1 |
| Automation Engineer | 3.9% | 4 |
| Test Architect | 1.9% | 2 |
| Senior Tester/ Sr. Test Engineer | 5.8% | 6 |
| Developer (who test their own or other's code) | 2.9% | 3 |
| Contractor full time | 1.9% | 2 |
| Contractor part time | 2.9% | 3 |
| Consultant full time | 4.9% | 5 |
| Consultant part time | 2.9% | 3 |
| Test Lead | 15.5% | 16 |
| QA Lead | 18.4% | 19 |

**Analysis**: Most popular job titles for people who test is QA, Test Engineer. 34% of respondents were Leads. Split almost evenly between Test and QA Leads. At least in the US, the QA role is disappearing and being more commonly replaced by QE or Tester/Test Engineer.

**2. Job Title - for Managers**

| | | |
|---|---|---|
| Test or QA Manager | 58.5% | 31 |
| Project Manager | 13.2% | 7 |
| Development Manager | 0% | 0 |
| Division Manager | 3.8% | 2 |
| QA Director | 20.8% | 11 |
| Director of Development | 3.8% | 2 |

**Analysis**: More than half of all respondents were Test or QA Managers. This makes many of the responses throughout the survey interesting from the perspective that many of the strategies, team dynamics and politics of testing questions were designed for staff QA/Testers but were answered by Leads and Managers who are very often on the front line of political battles and must defend strategy and time estimates. In total, two-thirds of the respondents to the survey were Leads, Managers, Directors.

**3. How many people are on your in-house onshore or offshore test team, not outsourced?**

| | | |
|---|---|---|
| 1 person team | 5.4% | 5 |
| 2 - 5 | 40.2% | 37 |
| 5 - 10 | 18.5% | 17 |
| More than 10 | 35.9% | 33 |

**Analysis**: The respondents to this survey were mainly distributed between smaller 2-5 person teams and more than 10 size teams. These respondents made up 76% of all respondents. This gives a nice balance to the survey with answers from large and small teams.

**4. What is the main reason you were hired?**

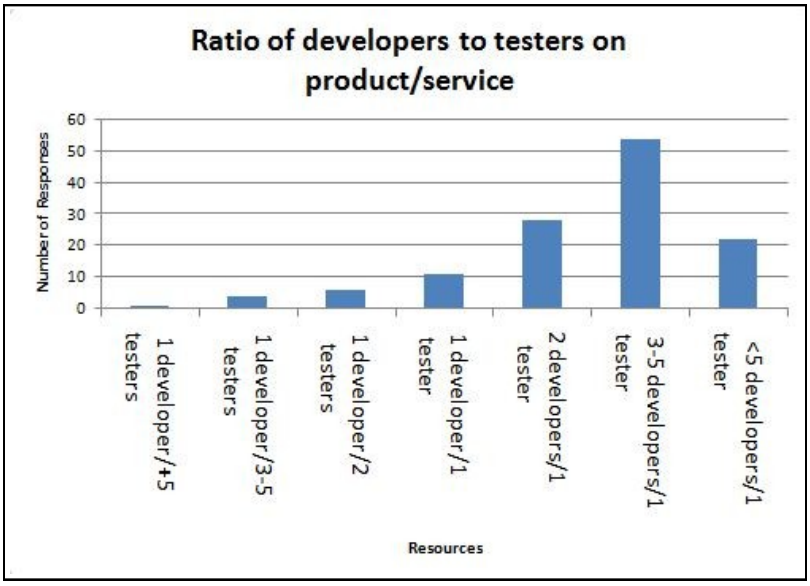
| | | |
|---|---|---|
| SME (subject matter expert/ domain knowledge) | 9.2% | 12 |
| Technology knowledge | 13.8% | 18 |
| QA/Testing knowledge | 56.2% | 73 |
| General job skills | 20.8% | 27 |

**Analysis**: This answer definitely surprised me. Conventional wisdom has it that most testers in the US and western Europe are hired with subject matter expertise. You hire bankers to test banking software! And that conventional wisdom has it that testers are hired in most LCC (low cost countries) where the US and Western Europe outsource/offshore are hired for technical knowledge. Wrong? That over 56% of respondents were hired because of their testing knowledge/skill is great and points to the value of testing knowledge, tester training and understanding quality theory and practices. But it is a surprise. I expected the number of people who test for their subject matter expertise to be the number 1 answer. It is number 4!

**5. Which do you feel is most important for testing?**

| | | |
|---|---|---|
| Technical software testing expertise | 18.6% | 24 |
| Domain/subject matter testing expertise, user focused tests. | 29.5% | 38 |
| QA/Testing knowledge | 51.9% | 67 |

**Analysis**: This answer follows the previous answer.




**6. What is the ratio of developers to testers on your product/service?**

**Analysis**: In all my consulting work, the most commonly asked question by directors is and non-technical manager is: *"what is the average ratio of developers to testers*?"

To answer the question, I quote a couple of published company ratios. Microsoft published at one point they had a 1:1 ratio of developers to testers. But, there is no industry standard. Just as banking software is not game software and operating system software is not an e-commerce site, there is no standard ratio. The needs are too different.

A general rule of thumb I use is, if you develop internal software (no external users), the ratio of developers to testers will be much higher. If you are a stock brokerage and your software is for your internal staff, stock brokers or fund managers, the ratio of testers to developers is very low. This is referred to as eating your own *dog food* (pardon the famous expression). This staff does not have much choice. Yet, in the same company, the teams that make, for example, the web portal for your customers to access their brokerage account information, buy and sell equities, will have a higher ratio of testers to developers. These customers, with the ability to move to another stock brokerage, need testing for ease of use/easy usability, testing to cut the cost of support calls as well as assess risk, assess security and performance and sometimes evaluate user experience against competitors. When you have more demanding customers, more competition requiring higher quality, your ratio will be more testers per developers.

Also affecting the ratio is the platform you support. Even with well automated projects, when you have to support multiple OSs, browsers, languages, mobile devices, devices with embedded systems- these situations will require greater numbers of testers. Consumer products generally have more testers than developers with reputation, support costs, warranty, PR, all potential risks to consumers, pushing the numbers of testers up.

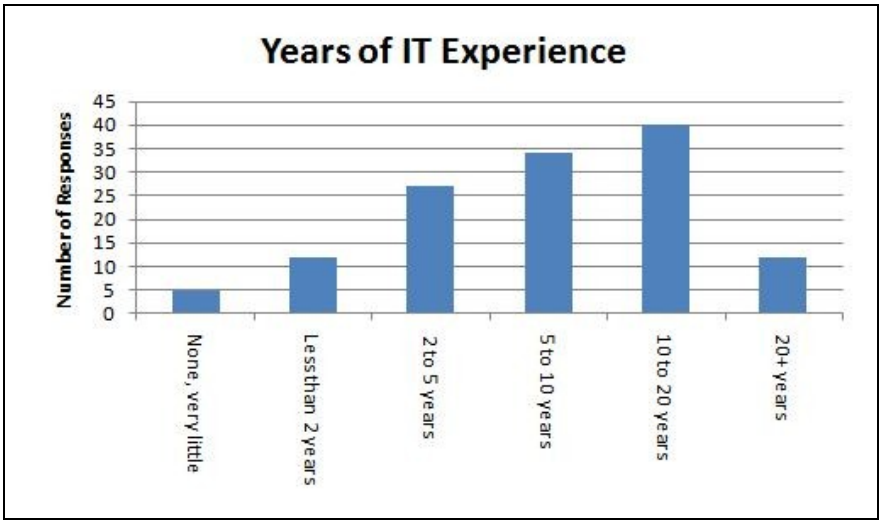
Games have high numbers of testers per developers due to the intense competition in their markets and big revenues.

For our respondents we have 63% with 2, 3, 4, or 5 developers to 1 tester. About 9% with a 1:1 ratio and a total of about 9% with multiples of testers to 1 developer. We also have 16.5% with *more than* 5 developers to 1 tester.

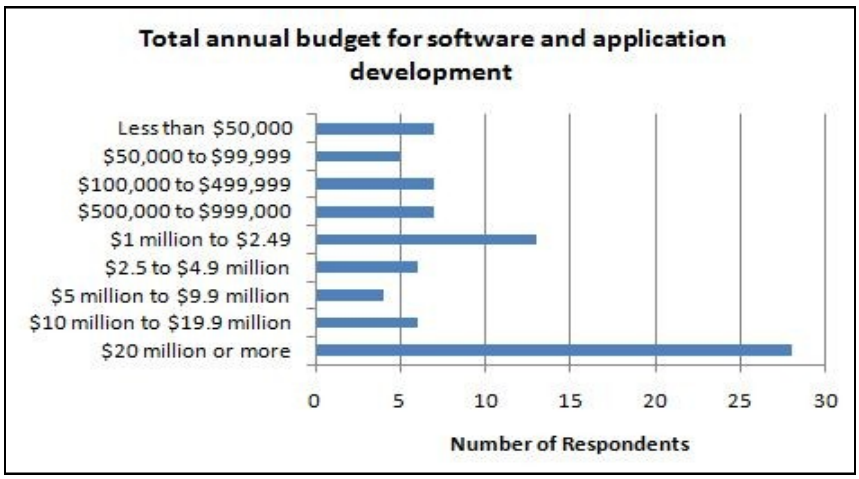
Looking through the list of companies responding, which will be kept confidential, there are consumer product companies, but the majority of companies make software that stays inside the company, or embedded software, hardware, ERP or B2B products. This majority of companies would have higher multiples of developers to testers.

**7. How many years of experience in IT do you have?**

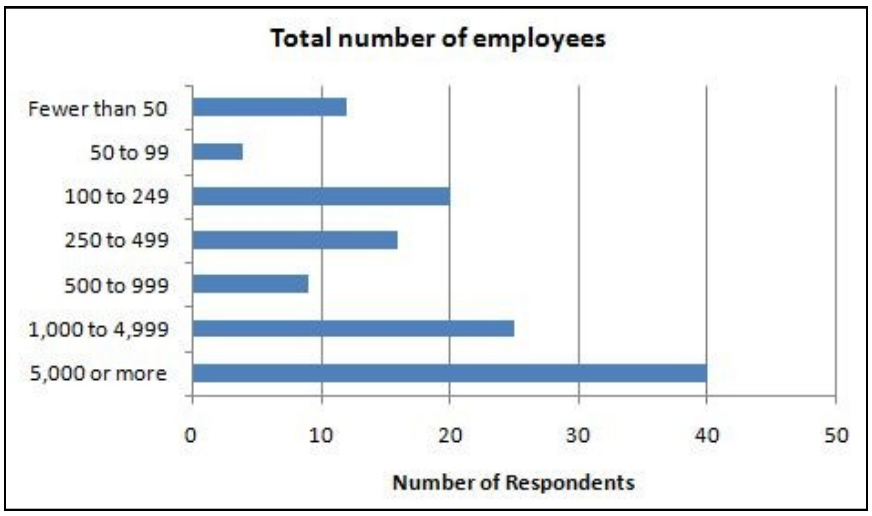
**Analysis**: Considering that two-thirds of the respondents are Leads/ Managers/Directors, this makes sense. This is a very experienced groups of survey participants.



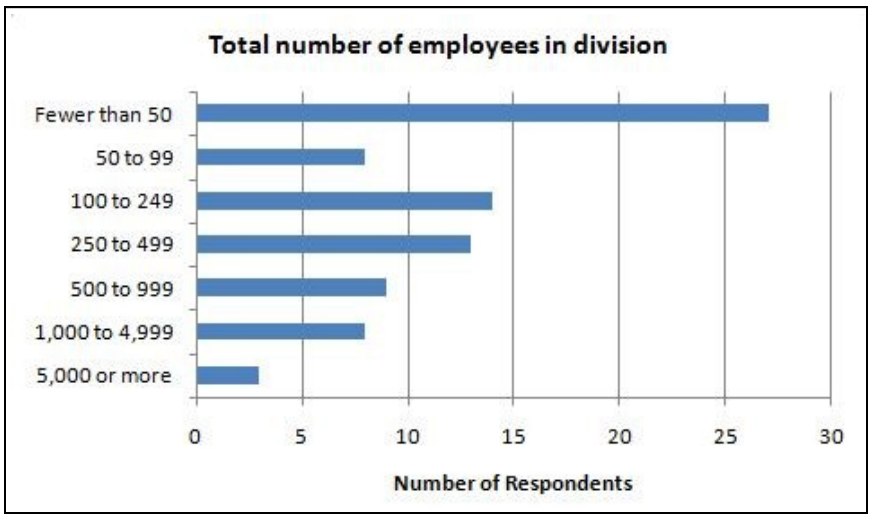
**8. What is the total annual budget for software and application development products and services at your company? (or at the companies to whom you consult)**




**9. What is the total number of employees in your company?**




**10. How many employees are in your division?**




**Analysis**: we have a good distribution of very large companies, big companies, mid-size and small companies. I often hear from software development teams: "Maybe they do that at big companies, but small companies like ours can't afford that many testers, can't afford to send our test team to training, can't afford an automation tool, can't take the time to use a test case manager, we're different…" When in fact, most software development teams, regardless of size of team, are very similar. Many of the questions in this survey point to the fact that regardless of company size, the problems and issues are the same.

**11. What programming languages are you comfortable with or use in your testing? (you can select multiple answers)**

| | | |
|---|---|---|
| Java | 38.3% | 44 |
| C, C++ | 34.8% | 40 |
| C# | 29.6% | 34 |
| PHP | 13% | 15 |
| Visual Basic | 40.9% | 47 |
| Perl | 14.8% | 17 |
| Python | 8.7% | 10 |
| JavaScript | 27% | 31 |
| Ruby | 6.1% | 7 |
| Delphi | 4.3% | 5 |
| Unix Shell Scripts | 18.3% | 21 |
| TCL | 7% | 8 |
| Other | 21.7% | 25 |

**Analysis**: This should burst some bubbles of conventional wisdom that a large percentage of testers are comfortable programming and very many in more than one language. Similar to conventional wisdom that test teams are hired for domain knowledge (wrong), that testers are technically proficient and seem to be moving to be much more technically proficient by the year is an important move for our industry.

**12. Do you feel prepared in your job skills to effectively test?**

| | | |
|---|---|---|
| Not prepared | 3.8% | 5 |
| Somewhat | 7.7% | 10 |
| Partially | 36.2% | 47 |
| Fully | 52.3% | 68 |

**13. Is your job technically challenging?**

| | | |
|---|---|---|
| Yes | 60.2% | 77 |
| Somewhat | 34.4% | 44 |
| No | 5.5% | 7 |

**14. Are you happy in your job?**

| | | |
|---|---|---|
| Yes | 60.6% | 77 |
| Somewhat | 35.4% | 45 |
| No | 3.9% | 5 |

**Analysis**: For the above answers, the overwhelming response is that people feel well prepared for their jobs and are technically challenged (2 of the 4 components for knowledge worker job satisfaction according to Peter Drucker). Overall, this level of job satisfaction is quite high, especially during years of the global economic slowdown during which this survey was taken. ■

## VIET NAM SCOPE

# Vietnam: Rice Country

*Rice and Vietnam are almost synonymous. Not only does the staple make up the country's culinary soul, it also helps keep its people in good health.*

**By Brian Letwin**

Many things make Vietnam a special place - a rich cultural history, kind people, natural beauty and, above all, great food. While almost all types of Western food are available in the country's major cities, Vietnam offers an abundance of delicious, low cost and healthy fare that can differ greatly by region. Thanks to their complex flavors and extremely fresh ingredients, some of the world's top chefs, such as Anthony Bourdain, Martin Yan and Gordon Ramsay, have made a point of highlighting this country's gastronomical delights.

Rice constitutes the backbone of Vietnamese cuisine. Vietnam is one of the top three global exporters of the staple, and the ingredient forms the foundation of almost every dish. The country is perhaps most famous for its noodle dishes, especially *pho* and *hu tieu*, both of which employ rice noodles to comprise the 'meat' of the soup. These soups are also highly customizable, as one can add garlic, chilis, fish sauce, hoisin sauce, lime, sprouts, and fresh herbs. I love a good hamburger, but with soup, I never feel weighed down after eating, allowing me to comfortably zip away on my motorbike with a satisfied pallet.

As with the rest of Asia, in Vietnam rice is also served in its traditional form with meat and vegetables, as a common dish called *com [pronounced: gom]*. When walking the streets during lunch hour, the sidewalks are lined with windowed food carts. Within each lies a plethora of choices to pair with the rice - from chicken to pork to fish - always served with veggies and soup.

Perhaps my favorite (or most romantic) aspect of food in Vietnam lies in its preparation. In western cities, there is little street life from 4am -5am (save for a few people returning home from a night on the town). But in Vietnam, as dawn breaks, one encounters elderly ladies starting their charcoal stoves, setting up huge tin pots and chopping endless quantities of garlic and vegetables. They offer not only a connection to my stomach but also to the past, as these dishes, and the ways they are prepared, have been an integral part of Vietnam's culture for hundreds of years.

The biggest change to my diet after moving to Vietnam has been a re-framing of the "fast food" concept. While there are more KFCs and Lotterias (a Korean version of McDonald's) popping up in the urban centers, fast food, for the most part, is comprised of soups and rice dishes. The effects of healthy food are evident in the population, as obesity hardly exists in Vietnam.

With seemingly limitless choices, it's no wonder that Vietnam has become a popular destination for culinary adventurers the world over. ■


LOGIGEAR MAGAZINE
APRIL 2012 | VOL VI | ISSUE 2

**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