

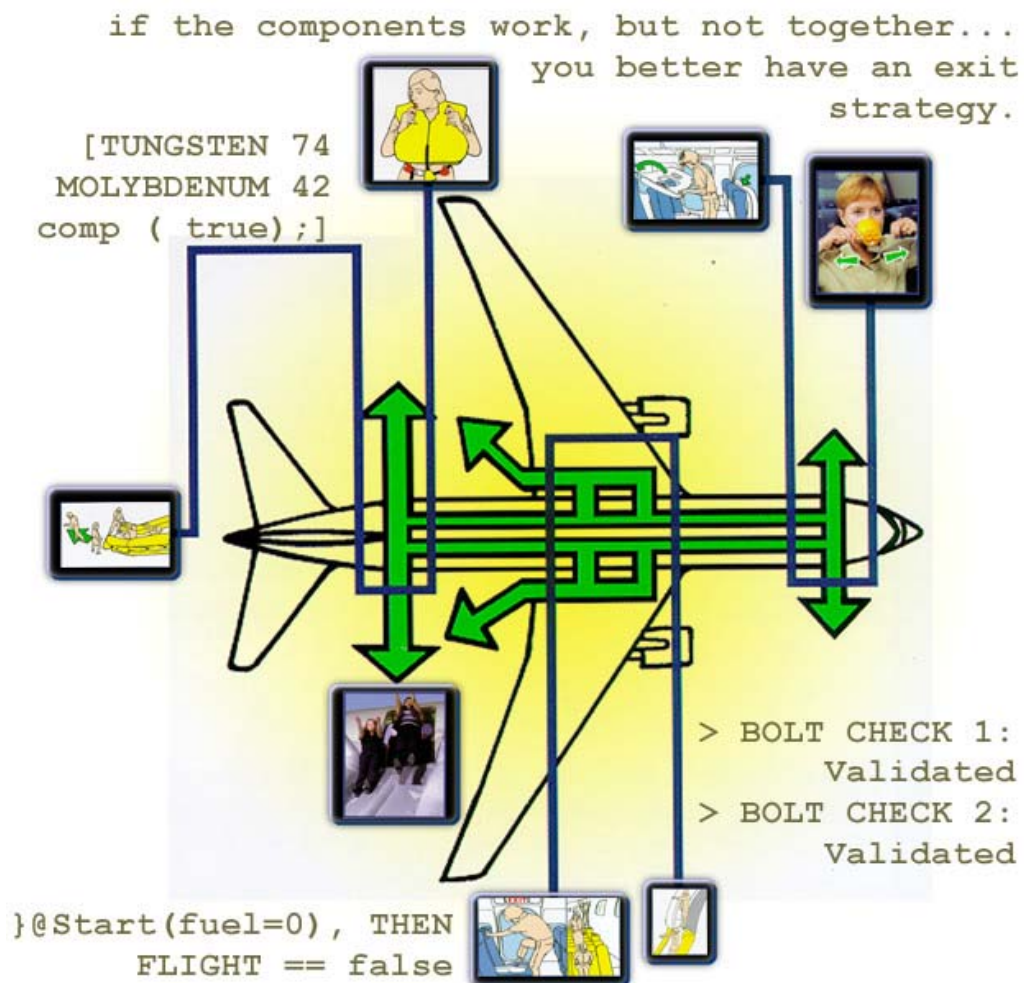
Why Unit Testing Alone Is Not a Strategy for Quality

***More unit testing can fix mistakes,
yet fail to prevent real problems***

*October, 2004
by John Michelsen
and Jason English*

Table of Contents

Intro: The Current Quality Situation	3
Defining Unit and “White Box” Testing	4
Solutions	5
1. If unit testing is your strategy, you have no strategy.	5
2. Testing tools need to support non-technical users.	5
3. Test at one layer and execute at another.	6
4. Test your systems under stress.	7
Conclusion	8
About the Authors	9



The Current Quality Situation

Unit testing as a quality strategy? It won't fly.

Yes, unit testing solves problems. Yes, developers should continue to practice it. But is unit testing still a relevant frontier for improving the quality of enterprise applications? Brand me a heretic, but I think it is only solving the small problems.

Testing during development is a quality discipline you need to enforce. Therefore, you've invested in training and tools for your programming staff. More unit tests are helping your development teams find structural bugs more efficiently. Churning through even more unit tests, and looking at charts that show how your byte code is 99.5% covered is reassuring. Then you realize while using the finished project at deployment time that the developers executed flawless code, but built an application that fails to meet business requirements in production.

In our experience, only **1 in 100 critical software failures that enterprises encounter in production could be prevented by better developer unit testing.**

Increasingly in the enterprise application development field, quality improvement initiatives have become oriented around either 1) testing the front-end UI of a nearly finished application to tell you "it's broken," or 2) promoting the practice of unit testing as a discipline reserved for developers. We believe these forces open up a serious quality gap.

This article addresses the second assumption: that code coverage (or "white box") developer unit testing can prevent most errors in production. Relying on unit testing alone to improve software quality is extremely dangerous when you are trying to uncover problems due to resources, business rules, logic errors, performance hang-ups, data quality, user behavior, security, long-term system stress and integration.

Defining Unit and “White Box” Testing

Who’s testing your unit tests?

Unit tests are basically coded programs that check to see if a specific piece of code or method passes or fails according to the developer's expectations. Understand we are not knocking unit testing, we do it avidly as a part of finding mistakes during our development too. Smart developers can usually find and correct bugs within their code through unit testing, and the advent of open source tools such as JUnit/Cactus have helped increase the awareness and common practice of responsible unit testing. Other solutions can help accelerate and manage the process of unit testing, which can yield some productivity improvements for developers.

Expecting developers to functionally test (while they are in the middle of developing the application, and coding unit tests, no less) is completely preposterous, yet real functional testing has become so difficult that the business side and QA team step out of testing until the application is near completion – when problems become the most costly to fix. Developers cannot be solely responsible for testing. They should be responsible for making their software testable by the rest of the team.

There are ideal situations for massively increasing your focus on unit testing. You would need "white box" testing capabilities, for instance, if you were a software product company publishing a toolkit or framework for other developers to use. But if you're like most development teams, you're not providing a toolkit, you are supporting an enterprise solution. You are building live applications that need to perform certain functions in production and support critical business operations. Maybe you are doing enough unit testing. Do you know that your application will actually work?

If you're delivering enterprise applications, there are compelling reasons why you shouldn't buy into the idea that "unit testing" alone will provide major quality improvements.

Solutions

1. If unit testing tools are your only strategy for quality, you have no strategy.

Developer-only tools, whether they are “white-box” solutions or manually integrated using open source technologies, can only test code in isolation from deployment conditions. Developers want to test more effectively, and they love a time-saving tool that will analyze and look at their code. At best these tools can automate only unit (code-level) tests, and they require a thorough programming knowledge.

Improving developer unit testing alone will never tell you if all the components of a complex distributed application will work for your business. It can help developers find mistakes in their code, but it won't tell you if they are misunderstanding a requirement. Whether they write manual tests or use a tool to help script unit tests, developers can still only test their code for the conditions they *think* will occur in the real world.

Compare it to testing an airplane. The unit-testing approach would be to take the plane apart and examine the stability and molecular structure of every part. This will tell you that the components of the plane *should* work once assembled. Useful at a root level, yes, but no substitute for a test flight. Would you fly in a plane without one?

The LISA difference: Yes, unit testing is useful and necessary, and LISA can extend these tools and offer some incredible ways for developers to both test instrument their custom apps and automate unit testing (without having to write or generate thousands of lines of test code). But LISA is about making sure the software works, and not just the code. LISA even allows you to globally attach and dynamically change data sets and randoms as they apply to a unit test, giving you the ability to thoroughly exercise developer code and components.

LISA is not lacking for relevancy in the developer's unit testing toolkit. LISA elegantly extends unit testing methods like JUnit/Ant, and developers can roll LISA test cases and launch them right within their unit tests and build scripts. And LISA offers developers a way to automate complex test creation, saving hundreds of lines of test code while providing far more detailed issue resolution information from developed systems than just “pass/fail.”

2. Make testing tools available to non-technical users. We've heard about “silver-bullet” no-code solutions that ultimately fail to allow business users to really test, or the blockbuster test-building consulting engagement that shields QA teams from manual test creation. The business side is still locked out of the testing process because it is simply too difficult to train non-developers to dig into a new programming language.

Developer-centered unit tests alone can basically give you several “observations” about the code, with little clue as to the relevancy. For instance, an observation you might find about our airplane:

```
> @START( fuel=0), THEN FLIGHT == false
```

[ok, so we just made a hard-to-read statement that I have to put fuel in the plane before I fly it...]

Get my meaning? Now what if among thousands of these observations, one stated that the plane could continue to fly without fuel? Unless you specifically remembered to exclude for this condition, your unit tests would have NO IDEA if that's valid or not.

So outside of manual testing, how can the business get involved? “No-code” testing solutions are great in theory, but you need to ensure that non-developers can actually test without coding in practice. If on day 1 of testing, your QA analysts can point and click on a UI, and on day 2 they find themselves maintaining incredibly complex test scripts, which break every time a change is made to an application or interface, keep looking. You need to be testing, not writing more code.

The LISA difference: If your users are smart enough to write and understand a complex set of business requirements, they are smart enough to start using LISA to test those same requirements within 1-2 days of training. Developers can pick up LISA even faster, and directly communicate their test cases to QA for additional modification. The more people in a team that use LISA, the more skill increases, as the same examples and test cases can be leveraged throughout the development, integration and deployment cycle.

And of course, you get iTKO's firm customer commitment, support, and services to make sure LISA allows "everyone to own quality" within your company.

3. Execute at one layer - and test at another. Today's enterprise applications are increasingly delivered as Service-Oriented Architectures (SOAs). An SOA is inherently made up of several interconnected applications, services and databases. Your users will execute on one layer – usually through a Web browser, but they will be interacting with other layers behind the user interface: web services, EJBs, legacy apps and databases.

The typical approach to testing an SOA today involves a developer writing a custom "test client" to give QA teams an interface to poke on and validate each one of these services. Teams end up throwing that test client away, so you are not only wasting a lot of development effort, you are only “testing the test clients” and not the services themselves.

When is the last time your developers had to build a distributed enterprise application “from scratch?” The usefulness of unit testing discipline diminishes when 1 in 10 hours of development time are spent creating brand new functionality -- where a given team of developers can actually apply best practices for unit testing.

The rest of the "relics" that you must leverage to make up your application may be developed by other teams, former employees, other consultants, or even hosted by other companies! These services are also "headless" applications that do not even have interfaces you can interact with. You may not have total control over these objects, but you need to analyze and thoroughly test these web services individually to ensure that they are behaving according to your business rules.

The LISA difference: For an enterprise application, this is the most compelling reason for using LISA. LISA becomes your pre-emptive client to these components, taking you out of the business of coding, creating and testing throwaway clients. While you are building a test case in LISA, you can simulate a browser dialogue and verify calls against a web service or database, then jump right into staging that test, then jump back and modify your test case **while your test is being staged without recompiling or modifying a single script.**

LISA lets you directly invoke multiple components in a "live interaction" way you simply have to experience to believe.

4. Fully verify your unit and functional tests under stress. Don't just test early. Stress test early, so you can still afford to correct a problem when a key element of your application breaks.

Under the real business conditions of load, variable use and continuous stress, most enterprise applications will fail, and finding the real source of a problem in a system under stress can be extremely difficult. Systems behave differently when they are put under load and interconnected with the rest of the infrastructure. Testing small units within your application will provide little confidence that your deployed solution will perform as expected in production. You need to do more than "check a box" on validating that every component of your system under test can handle the actual conditions in which it will be implemented, and you need to do so early enough in development to make corrections within your project timeline.

How can you ensure that the application actually fulfills a complex workflow and meets your business requirement? You will need to roll the logic you validated in your unit, functional and regression tests directly into load tests and continuous monitoring. For instance, you might find that a test case that always worked with one user will fail with 100 real concurrent users, as two of the users get each other's account information back from the middle tier.

To bring back the airplane analogy, once you've ensured that the plane can fly, you better be confident it can still fly in a storm, when it is full of real people and luggage shifting around within it, or you are in serious trouble.

The LISA difference: In the same application you use to create unit functional and regression tests, LISA offers a non-assumptive load test, with full functional verification of every simulated user. Within one LISA test case, both developers and QA teams can directly instrument and test multiple components of an SOA. How else can you ensure that the application actually fulfills a complex workflow and meets your business requirement? No other tool allows you to roll the logic you validated in your unit, functional and regression tests directly into load tests and continuous monitoring.

Conclusion

What else have we learned in 10 years of enterprise development?

We need to face the issue that when developers alone are conducting testing activities, even using a tool that automates test script generation, they can only test for the conditions they expect to happen in the real world. We know that manual testing and unit testing cannot possibly uncover all the conditions that will occur in a complex (SOA) business application.

SOAs offer the promise of new business capabilities, as now you can leverage legacy systems, new components and even third-party applications, but each point of connection or integration becomes another potential point of failure. Before deploying, you need more functional testing, beyond just testing your own code. You need to thoroughly test all of those "headless" applications and components, many of which may not be under your direct control.

Unit testing will not prevent your most critical software failures. Testing at the unit level is useful for allowing developers to find and correct their own mistakes. These mistakes can be costly to fix, but it is the misunderstandings -- the missed business requirements and failures under stress -- that can kill.

Everyone should own quality. Laying most of the burden on developer testing as a cure for software failures is a cop-out. Business users and QA teams need to stay involved in functional testing during development if they are going to be responsible for the application meeting real-world business requirements. Getting there will take a new level of discipline and cooperation across the entire team. But when you can get everyone to own quality throughout the development process, you can finally deliver with confidence.

And that should help you sleep on the plane.

About iTKO

iTKO is a software firm tightly focused on providing development solutions that support their mission to allow the entire organization to own quality. Operating from a suburb of Dallas, TX, iTKO has a diverse customer base, ranging from some of the world's largest financial institutions and software design firms, to individual developers and QA testers. iTKO's products help enterprises deliver maximum quality with minimum effort and cost. For more information on iTKO and LISA automated testing software, visit our site at <http://www.itko.com>.

About the Authors

John Michelsen is CEO and a co-founder of iTKO, Inc., an automated software testing company. John has more than 15 years of high-level enterprise development experience, as a chief architect of development teams and as an executive in designing, developing, and managing large-scale, object-oriented solutions in traditional and network architectures. He is the chief architect of iTKO's LISA automated testing product and a leading industry advocate for software quality.

Before forming iTKO, Michelsen was Director of Development at Trilogy Inc., and VP of Development at AGENCY.COM. He has been titled Chief Technical Architect at companies like Raima, Sabre, and Xerox while performing as a consultant. Through work with clients like Cendant Financial, Microsoft, American Airlines, Union Pacific and Nielsen Market Research, John has deployed solutions using technologies from the mainframe to the handheld device.

Over the years, John and his teams have enjoyed tremendous delivery success. At Sabre, Michelsen served as the Chief Technical Architect of the first real-time Internet reservations application. He served as lead architect for the www.enrononline.com trading marketplace development which processed over a billion dollars of trading activity daily. John consulted Microsoft to help establish itself in the enterprise software market. These are just a few of the client solutions and relationships which are still in operation today -- a testament to his technology vision and customer commitment.

Jason English is Chief Marketing Officer of iTKO, Inc. As a software workflow designer and strategic marketer, Jason has more than 13 years of experience creating user interfaces, gathering customer requirements, and executing marketing plans for technology companies such as HP, IBM, EDS, Sun, Adaptec, Motorola and Sprint. Jason served as Executive Producer of i2's interactive consulting unit and as a Media Architect there, where he was responsible for creating all of i2's outbound messaging, as well as marketing services and working directly with clients to create easy-to-learn front ends to B2B collaboration systems. Prior to that he served as one of the first "Information Architects" at Agency.com, and previously wrote, scored and designed several internationally released computer games in addition to conventional print advertising and television commercials. Jason has a BA in Journalism and Communications from Baylor University.

Terms of Use

© 2004, iTKO, Inc. This whitepaper can be distributed and shared between colleagues on a personal basis only. Any other business use, reprint, rewrite or document posting must be approved in writing by iTKO. Inquiries should be sent to info@itko.com. LISA™ is a registered trademark of iTKO, Inc. Document Owner: Jason English, Chief Marketing Officer, iTKO, Inc.