The Dangers of Use Cases Employed as Test Cases

Bernie Berger

This document is intended to provide background support and additional information to the slide presentation at STARWest 2001. I don't consider this a complete paper without the associated MS PowerPoint slides and presentation.

Use cases can often help jump-start a testing effort, but there are serious side effects when testers use (only) usecases as a testing guide.

What are Use Cases?

Use cases are a relatively new method of documenting a software program's actions. It's a style of functional requirement document – an organized list of scenarios that a user or system might perform while navigating through an application. According to the *Rational Unified Process*, "A use case defines a set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor".

What's so good about Use Cases?

Use Cases have gained popularity over the last few years as a method of organizing and documenting a software system's functions from the user perspective. They support an iterative development lifecycle, and are part of the Rational Unified Process. (For more information about use cases, please see www.rational.com/rup)

What are some of their problems?

There are problems inherent in any documentation method (including traditional functional requirements documents), and use cases are no different. Some general problems to be aware of include:

- They might be incomplete
- Each case not describing enough detail of use
- Not enough of them, missing entire areas of functionality
- They might be inaccurate
- They might not have been reviewed
- They might not updated when requirements changed
- They might be ambiguous

What are some problems with using use cases to test software?

In addition to the generic problems listed above, there are specific challenges when testers use use-cases to test.

They are only useful for certain types of testing:

- User Acceptance Testing
- Positive "business as usual" Functional testing
- Manual black-box (some)
- Scripted automation -Automated testing – Use cases would be able to help you automate test scripts. It seems as though use cases are well suited to generate (even record) a suite of positive regression scripts. The use case helps you decide which are the common scenarios to record automated scripts for regression testing.

They could also help types of testing that overlap (or are co-defined) with positive functional testing:

- Smoke testing
- Sanity testing
- Regression testing
- Ad-hoc testing

What kind of bugs would not be discovered?

There are many types of bugs that you would NEVER find by "testing" a program by following its use cases. Some of them include:

System testing – Could you find a memory leak bug (not by accident) by following a use case?

Integration testing – A use case for a financial trading system states that a buy order for a security can be good for one hour, one day, or until canceled. A tester (actor=buyer) following this use case might enter buy orders, then advance the system time/date to see if the order is removed from the system. But what the tester might not do (since it might not be explicitly explained in the use case) is check if the order appears and goes away on the seller's workstation, or in the admin screen that summarizes all open orders in the system.

Performance testing – Would you be able to discover a defect for an advanced search that would take too long to return due to an inefficient database schema?

Load testing – Would you be able to test the load boundaries by following use cases?

Software Compatibility testing --Browser/OS: Drop-down lists or scrollbars are improperly formatted on Netscape 4.72. Did the use case directly help you find this bug?

Hardware Compatibility testing – drivers, printers, processors… Your app doesn't print with older printers; use cases won't help you find this bug.

Exploratory testing - Exploratory testing practices state that you learn as you test, and that your next test depends on what your last test was. It is a heuristic search for bugs not to be constrained by use case documents.

Are there other concerns?

- Because its intended design wasn't meant to find bugs, you probably won't find many by using use cases. It would be poor test case design, for one reason, it doesn't use challenging input.

- They promote an impression that testing can be done by anyone, without needing any skill or talent.

- They promote an impression that the goal of testing is to make sure the software works, rather than to find bugs.

Conclusion

Use cases are a great way to organize and document a software system's functionality from the user's perspective. However, they have limited uses for testers. They are great vehicles to accomplish some tasks, and not-so-great for others. Understand what you're trying to accomplish by testing before deciding if use cases can help – and be cognizant of the challenges they present. They *are* useful to testers, but not for every situation.

Use cases are a relatively new method of documenting a software program's actions, and therefore the use cases you're looking at might be the product of a new technical writer who never before wrote use cases. The usecase might be incomplete. It's possible, actually likely, that the use cases didn't go through a formal (or even informal) inspection. This is because Use Cases are typically used on Internet type projects, with Internet type staff, on an Internet type schedule. How many internet projects do you know of that have the "luxury" of reviewing work products?

Is it possible that this is the first time some people of management of the staff ever saw usecases? They, the ones with the control over what and how things should be developed, might not have before used usecases, and therefore might not know what to expect from them…..

So they might expect, put pressure on the test team and test manager to use the use cases to test the product. "this is what it's supposed to do, make sure it does that".

Some things to look for as far as omissions from test cases: what we call negative tests. Example: It's quite possible that the author put in everything that's supposed to happen when a user would do a certain this or that to the application. That's what it's supposed to do, so that's what is in there. What about the things that the software is not supposed to do? Sure, maybe there's some error handling in there. But there's more to negative testing than error handling. How are we to determine equivilance class and partitioning?

There are two types of completion/incompletion. A particular use case might be incomplete, by that I mean that there are elements of a single action that are missing. For example, what if the user does this in a certain circumstance, but it might not cover if the user does that. The other kind of incompletion is not when a single usecase is missing information, but when whole usecases are missing from the use case document. For mple

Use cases, if they are complete, and have been reviewed, (these are big iffs) are great for usability testing and positive, and perhaps negative functional testing. Is functional testing enough? Aren't there dozens of types of testing besides functional testing? Just think about what you might have listed on your resume: To name a few types of testing that aren't touched by usecases:

System testing – How do use cases specify how to test aganst system resources? Could you find a memory leak bug by following a use case?

Integration testing -- , a usecase might tell you how data should be entered into a screen, and what error messages might come up. But what happens to this data?

Presumably it's supposed to be stored in a database on the back end. Perhaps it is supposed to be displayed on another screen within your system (which the usecase would (might) specify) but perhaps it's also supposed to be displayed on other systems connected to your network. And what happens when the information that you entered becomes invalid…should it automatically be removed from your screen? Should it also be removed from others' screens? Under ehat conditions? What entitlements or user rights are suppoed to be active? If you find a bug like this, it's likely that you found it not by following a use case. These are system-wide tests.

Performance testing – Would you be able to discover a bottleneck defect for an advanced search that would take too long to return? How would following a usecase help you find a defect with the database scema?

Load testing – would you be able to test the load boundaries by following usecases?

Software Compatibility testing-Browser/OS: Drop-down lists or scrollbars are improperly formatted on Netscape 4.x When did you learn about this? Probably late in the test cycle, no? Did the use case help you find this bug?

Hardware Compatibility testing – drivers, printers, processors… Your app doesn't print with older printers, use cases won't help you find this bug.

*Automated testing – Use cases would be able to help you automate test scripts. it seems as though use cases are well suited to generate (even record them) a suite of positive regression scrpts. (the use case help you decide which
Stress testing
BlackBox/WhiteBox
Smoke Testing
Sanity Testing
Rapid Evaluation Testing
Embedded software testing
Communication testing
End to end testing
Ad hoc testing
Exploratory testing

How do any of these testing types are addressed by usecases? With a few possible exceptions, they aren't. Exceptions might be black box/smoke/sanity testing, since these are closely related to functional tests.

A resulting problem is that

Since the use cases only cover positive functional tests, and they might cover negative functional tests, you might release the project without enough testing on it.
The inexpericenced project manager might not understand – here are the use cases. I want the programmers to program it like this, and have the QA folks test it. This is all they need to do. Just make sure it works.   The usecase as testcase model is fodder for inexperienced project managers to expect QA to "just make sure it works".
Even worse, the test team themselves might get fooled into believing that just because the program does what the use case says, it's OK. Usecase as testcase model promotes the reduction of creative test design.

Bernie Berger

Over the past thirteen years, Bernie Berger has been a Computer Operator, Software Tester, QA Supervisor, Test Consultant, and QA Manager. He recently founded Test Assured, Inc., a boutique consulting firm specializing in software testing and quality improvement for the financial sector. Bernie is a QAI - Certified Quality Analyst and Certified Software Test Engineer, and is a special contributor to stickyminds.com.

Bernie can be reached at bernie_berger_qa@yahoo.com