What is Regression Testing?

Written by: Tom Huston

Introduction

Whenever developers change or modify their software, even a small tweak can have unexpected consequences. Testing existing software applications to make sure that a change or addition hasn't broken any existing functionality is called regression testing. Its purpose is to catch bugs that may have been accidentally introduced into a new build or release candidate, and to ensure that previously eradicated bugs continue to stay dead. By re-running testing scenarios that were originally scripted when known problems were first fixed, you can make sure that any new changes to an application haven't resulted in a regression, or caused components that formerly worked to fail. Such tests can be performed manually on small projects, but in most cases repeating a suite of tests each time an update is made is too time-consuming and complicated to consider, so automated testing is typically required.

Understanding Regression Tests

Some software development teams try to get by without performing regular regression tests, opting to test essential functions just once to make sure they work and, if they check out, proceeding with the hopeful assumption that those functions will still work unless they're directly modified again. In a way, this makes sense: it's natural to want to simply make a change, test it, and move on. Performing functional tests or highly specific unit tests to determine that a new software component works as it should has been called "non-regression testing" by Doug Hoffman and others. But it can be relatively easy to find a specific problem when you're looking for it; what's harder is catching all the ones you don't expect.

Again, it's important for developers and testers to always bear in mind that even small, seemingly insignificant alterations to an application's source code can ripple outward in surprising ways, breaking functions that seem completely unrelated to the new modification. When you run regression tests, you're checking to make sure that your modification not only behaves as you want it to, but that it also hasn't inadvertently caused problems in functions that had otherwise worked correctly when previously tested.

Fortunately for the would-be regression tester, on any given project your regression test libraries can be built from the existing test cases developed from day one. Functional tests, unit tests, integration tests, and build verification tests—anything that has successfully verified, throughout the development process, that various components work as intended—can all be incorporated into a regression testing suite, and "regression tests," per se, don't necessarily need to be written. Each time you modify your source code, you can simply re-run the potentially relevant tests to ensure that they continue to pass. Naturally, over the course of a complex development project, those test cases—and the various functions and processes that they attempt to check—can number in the thousands, making the use of automated testing software mandatory for full-scale regression tests.

To reduce the stress levels of programming teams everywhere, various automated testing programs that specialize in regression tests now make it relatively easy, with a few clicks of a mouse, to establish sets of testing parameters and to check new iterations of code against previous software baselines, or control states, highlighting inconsistencies in testing logs and specifying exactly where

an unexpected function broke and why. You may not have the bandwidth or time to let your software re-run *every* test, checking the entire application for potential errors, but you'll dramatically exceed what you'd be able to check manually. Indeed, when it comes to automated regression-testing tools, sometimes it almost seems too easy.

And if you're not careful, that can be a problem.

The Limitations of Automation

As with most forms of automated testing, setting a regression-testing program on autopilot is not a surefire solution, and some conscious oversight and input is generally still needed to ensure that your tests catch all the bugs they should. When you have the exact same suite of tests running repeatedly, night after night, the testing process itself can become static. Over time, developers may learn how to pass a fixed library of tests, and then your standard array of regression tests can inadvertently end up not testing much of anything at all.

In some ways, making sure that your software continues to adhere to requirements specifications as you develop it is like clearing a path through a minefield. To proceed safely through enemy territory, professional minesweepers don't need to clear the entire field; they just need to clear a single path, set up defined boundaries, and guard that path. Once enough soldiers are there guarding it—soldiers who, in this analogy, represent your regression tests—it's unlikely that anyone's going to slip a new mine onto it. However, this continually clear path says nothing of all the other mines that might still be out there, old or new, just waiting for aimless civilians to step on them.

Likewise, if your regression testing becomes too automated and rote, the whole point of doing it can backfire. You can end up guaranteeing a clear software-development trajectory for yourself and your dev team while unwittingly ignoring vast swaths of the application, letting your end users stumble upon undetected glitches at their own peril. Walking along a single path of least resistance is, of course, easier than stopping to sweep the entire application after each new step, but it's worth the effort to take your regression testing all the way by frequently scanning a little further afield. And that often just means complementing your automation with some good old-fashioned manual tests.

The Future of Regression Testing

For regression testing to be effective, it needs to be seen as one part of a comprehensive testing methodology that is cost-effective and efficient while still incorporating enough variety—such as well-designed frontend UI automated tests alongside targeted unit testing, based on smart risk prioritization—to prevent any aspects of your software applications from going unchecked. These days, many Agile work environments employing workflow practices such as XP (Extreme Programming), RUP (Rational Unified Process), or Scrum appreciate regression testing as an essential aspect of a dynamic, iterative development and deployment schedule.

But no matter what software development and quality-assurance process your organization uses, if you take the time to put in enough careful planning up front, crafting a clear and diverse testing strategy with automated regression testing at its core, you can help prevent projects from going over budget, keep your team on track, and, most importantly, prevent unexpected bugs from damaging your products and your company's bottom line.

Article from: http://smartbear.com/all-resources/articles/what-is-regression-testing/