REGRESSION TESTING: A BEGINNER'S GUIDE

In the software industry, there's a well-known problem called regression. In short, it's when you fix a problem only to discover that there are new problems created by an update. The rise of big data makes regression an even greater problem for organizations.

We all remember the Intel Meltdown and Spectre problem at the beginning of the year, right? Intel and AMD's attempt to patch away the problems led to some disastrous results.

Regression testing is the answer to this common problem. What is regression testing and how can it help your business? Read on to learn more!

Types of Software Testing

To understand regression testing, let's first look at the two categories software testing falls into.

- First, there's whitebox testing. Whitebox focuses on the internal mechanisms of the software. It's used for verifying different software applications.
- Blackbox testing is the second category. It ignored the internal mechanisms and instead focuses on the generated output of the software. You might see blackbox referred to as functional testing, as it focuses on the functions of the software tested.

Regression testing lies in the category of blackbox testing.

What Is Regression Testing?

This type of testing looks for regressions in software. A regression, as stated above, is when a problem arises after an action that involves software. The action could be in the form of an update, code modification, or patching.

Regression testing makes sure that one of these changes doesn't impact other areas of the software. It also ensures that there are no bugs in the added code.

It's not meant to test the action itself. It won't ensure that the update works as intended, only that it doesn't have unforeseen consequences.

The testing takes place after the action but before it's rolled out on an organizational level. The timing of the test depends on the type of test used, which we'll get into in a moment.

Do Small Changes Require Testing?

It's important to note that some people believe that regression testing isn't necessary for small changes. Others believe that only essential functions need testing. As long as they work, does the other stuff matter?

Absolutely. First, regression testing allows you to make future changes with confidence. You're able to narrow down problems if you know what changes in the past affected certain features of your software.

Also, these secondary functions might be the only reason someone uses your software. Sure, the primary functions work, but secondary functions are often as important.

Instead of having to fix problems that might arise later, regression testing allows you to proactively spot them. This saves you time and money, as you don't have to pay your development team for extra work.

Minor software updates can lead to serious problems. In 1990, AT&T spent millions to restore service after a minor update brought down their network. Learn from their mistake. Don't assume that a minor change can't wreak havoc on your organization.

What Types of Changes Require Regression Testing?

There are four main changes that require regression testing.

1. **New Functions** - The most common reason to run regression tests is adding a new function to software. Anytime you add new code, it has to be fully compatible with the old. Developers put the new code in without considering its compatibility. They trust regression testing to spot problems.
2. **New Integration** - Another common reason to run a regression test is when you integrate one product with another. Testing ensures that the two products work as intended after integration.
3. **Revised Functions** - Sometimes, developers need to correct software for functionality purposes. When this happens, they might revise existing features while discarding or editing others. Sometimes this causes unforeseen problems. Regression testing ensures the functionality of the rest of the product.
4. **Bug Fixes** - The fourth type of change is when developers fix pre-existing bugs. Sometimes, fixing one bug creates another because developers have to change the source code.

Regression testing makes sure that these changes didn't break or change the program.

What Are the Types of Regression Testing?

Regression testing works by executing pieces of code repeatedly and seeing what the results are. There are different techniques for executing the code based on what the new code might affect.

Here are the most common types of testing.

Corrective Regression Testing

If there are no changes to the product's specification, corrective testing is the way to go. This type of testing uses pre-existing test cases, making it fast and efficient.

Retest-All Regression Testing

This type retests all the aspects of a product and reuses all test cases, including situations when changes and modifications aren't made. Retesting is time-consuming, so it's not the best method for small changes introduced to existing products.

Unit Regression

In this method, testers treat code as a single unit. Unit regression happens before integration and blocks interactions that go beyond the code.

Partial Regression

This type takes the single unit of code tested in unit regression and integrates it with the pre-existing code. Testers see whether the change in coding allows the software to function as intended.

Complete Regression

The pre-existing code and new code integrate so that testers can see the whole picture. Developers gain a complete view of the system at large. Complete regression is necessary if the added or modified code has a major impact on the roots of the previous code.

You'll also need this type of testing if you're making changes to many parts of the code.

How Can You Perform Regression Testing?

Once you've written the modified or new code, there are a few steps you'll need to take.

First, perform a smoke and sanity test. No, this isn't a trip to the psychiatrist's office, although you might need it if the new code blows up your software.

Instead, these tests confirm your program's stability before regression testing. You aren't looking for bugs or glitches at this point. You're making sure that your system is stable so it doesn't crash once you start the test.

Next, you'll analyze the code requirements. Most of the time, the last-minute alteration of codes cause reported bugs. You'll want to analyze your new build before adding it to make sure it meets the users' requirements.

Look at test cases to determine which of the three types of regression testing you'll need to perform. You need to find out if you'll need partial, unit, or complete regression testing before you

begin.

Finally, examine the behavior of your organization and determine when your software gets used the least. Schedule the test for this time and test the software thoroughly before releasing it.

You can also use AI to run tests on your programs as well, depending on your company's needs.

Challenges of Regression Testing

Let's look at two of the common challenges companies face when it comes to regression testing and our answers to them.

Regression Testing Is Too Expensive

When you have to test a program numerous times during a build or update, it's difficult to justify the cost upper-management sometimes. Why should the company test something that they've already tested before?

This concern is a legitimate one, but one that's easy to allay. The cost of an important piece of software failing more than outweighs the cost of regression testing, especially if the software impacts your ability to communicate with customers. Downtime doesn't just hurt your sales, but your reputation as well.

Every problem that testing catches is a potential disaster averted.

Optimization and Maintenance

Whenever you make changes, you might need to modify existing regression test suites. This could require you to add, remove, or edit existing test cases. You'll need to do so in the allocated time for regression testing, taking up time and placing added stress on development teams.

This problem stops many companies from running regression tests, but it shouldn't. The time spent optimizing and maintaining suites pales in comparison to the time needed to find and fix software problems after an update.

It's not only your development team that suffers. The rest of your team might lose access to valuable tools that affect customer service. A little bit of extra time and energy now could save you money and stress later on.