eBook

Unit Regression Tests

A new type of test created by
Diffblue Cover

diffblue
AI for Code

What's wrong with traditional unit testing?

Unit tests, which verify the behavior of individual building blocks of code, are expensive. They are time-consuming for developers to write, they can be resource-intensive to maintain, and a critical mass of them is needed before they add much value. But they are also expensive to not create: if a major bug slips through to production, it can cost the business untold sums in lost customers, revenue and reputational damage.

Traditionally, there's been no way around this expense, so development teams have accepted a trade-off: under pressure to deliver new feature code, developers typically only write the unit tests that cover the business logic they think is most critical. They save time, but risk missing any number of less obvious, but still critical test cases.

A changing approach to unit testing, powered by AI, has made it easier than ever for development teams to get more of the tests they need. ***Unit regression tests*** are a new category of tests created by Diffblue Cover, a tool developed by the team of world-leading experts in software verification at the University of Oxford spin-out Diffblue.

By describing the historical behavior of your code, these unit regression tests track both intended and unintended changes in the behavior of code over time. Their strength is in their numbers and the speed at which AI creates them—hundreds of times faster than the equivalents could be written by people. But what are they, how else do they differ from traditional unit tests, and what benefits do they offer?

This eBook provides a rundown of the basics of *unit regression tests* written by Diffblue Cover:

- **What are unit regression tests?**
- **How are unit regression tests created?**
- **The financial benefits of unit regression tests**
- **How to get started with unit regression tests**

What are unit regression tests?

Traditional functional regression testing and unit testing aren't topics that are typically discussed together, but they ultimately aim to achieve the same goal in two different ways, with varying levels of success.

Regression tests (typically an automated functional test) aim to verify the consistency of functionality from release to release—usually investment is focused on important use cases. They typically run late in the pipeline, as part of final verification of the program, and are usually end-to-end functional tests, which require a realistical test environment, including dependencies like databases, APIs, etc. For these reasons, traditional regression tests are typically slow, expensive and (as a black box) ineffective at helping find where the unintended behavior has been introduced.

Traditional unit tests, on the other hand, run as early as possible in the development cycle and are designed to pinpoint errors in a single module. They have to run quickly so they can be used by the developer as part of the code-build-test-repeat cycle without impacting productivity. Having a full set of dependencies (such as databases, APIs, etc.) isn't practical for fast-running tests that run often. Dependencies are mocked—stubbed out with code that

returns test data, for example, returning simulated data instead of making an actual database query.

Upon closer inspection, it becomes clear that one of the main purposes of unit tests is to also find regressions. Once a unit test has been authored and the code committed, the unit test will forever provide a benchmark to which future commits can be judged. Developers run unit tests periodically to see if something that previously worked has broken or changed. This, however, only starts to deliver value if a critical mass of code is exercised with a complementary critical mass of unit tests. This is defined as code coverage, expressed as a percent of how much code is exercised by the unit test.

Having a small number of targeted unit tests can prevent some key issues, typically for high risk business logic that the developer has deemed valuable enough to write unit tests for. Unit tests don't provide any protection outside of the code that they cover, and this is why most organizations fail to see regression benefits from their unit tests without investing significant human resources in re-visiting existing code to implement unit tests.

Unit regression tests that are written and maintained automatically by Diffblue Cover, on the other hand, exist in volume by default, as will be explained in the next section. This is why they can quickly and efficiently allow developers to find changes in the behavior of their code, and even in edge and corner cases. Having a wide array of unit tests reduces risk and associated cost, so developers can have more confidence that the changes they make won't break the pipeline.

How are unit regression tests created?

The primary difference between traditional unit tests and unit regression tests is that unit regression tests are automatically created using Diffblue Cover. Diffblue Cover's AI engine can quickly write a suite of unit tests that are derived from existing code, so they reflect the current functionality of the program. This primary map of the code is used to test new commits, for example, to see what changed.

Because unit regression tests will only be created en masse, the individual tests themselves matter less than their collective capability as a net for catching regressions. The breadth and depth of tests that can be generated by Diffblue Cover quickly encompasses a wide variety of scenarios, including edge cases, corner cases and simpler boilerplate code that might have been missed (either intentionally due to cost, or accidentally) by a human.

What do unit regression tests look like?

Tests created by Diffblue Cover always compile, run quickly, and are easy to understand. Here's an example of a test created by Diffblue Cover for JUnit.

This unit of code checks the current balance and open status of a bank account:

```
public void takeFromBalance(final long amount) throws
AccountException {
    if (getAccountState() != AccountState.OPEN) {
        throw new AccountException("Cannot continue,
account is closed.");
    }
    if (currentBalance + amount < 0) {
        throw new AccountException("Not enough funds");
    }
    currentBalance -= amount;
}
```

A test for this code that was automatically created by Diffblue Cover is below:

```
@Test
public void takeFromBalanceTest() throws AccountException
{
    // Arrange
    Account account = new Account(1234567890L, new
    Client("Bob"), 10L);

    // Act
    account.takeFromBalance(10L);

    // Assert
    assertEquals(0L, account.getCurrentBalance());
}
```

In this test, the `Account` object has been initialized with real-world value of an account number, the client name and the account current balance.

In `'// Act'`, the method `takeFromBalance` is being called with 10, which will exercise the method logic.

In `'// Assert'`, the test is asserting that if you take 10 away from the Account balance of 10, you will be left with 0.

How this protects you against a regression

This test demonstrates how the method is meant to behave. If the method functionality were to be changed in an unintended way, this test would fail: For example, a trivial but common mistake is using the wrong operator; just changing the operator in "currentBalance += amount" would cause the test to fail, alerting the developer to a regression.

The financial benefits of unit regression tests

Shift Left Testing

The cost of fixing bugs increases drastically when they are identified in later stages of the SDLC: It's 30 times more expensive to fix a bug that's detected in production or post-release, compared to one found in the requirements/architecture stage. Because AI-generated unit regression tests are small and plentiful, they compile and run quickly to find precisely where the errors are very early in the pipeline, well before traditional regression tests run.

This makes it easier to find and fix mistakes when it is the cheapest and easiest to do so (before the developer has moved on to the next task) and also drives ownership and accountability for quality.

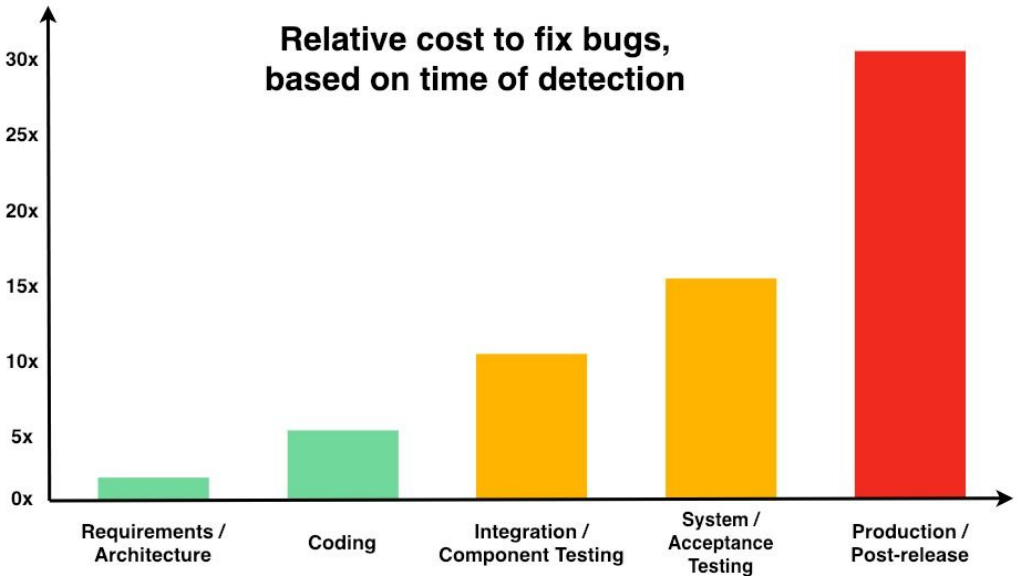


Image Source: The Exponential Cost of Fixing Bugs on DeepSource.io, with data from NIST

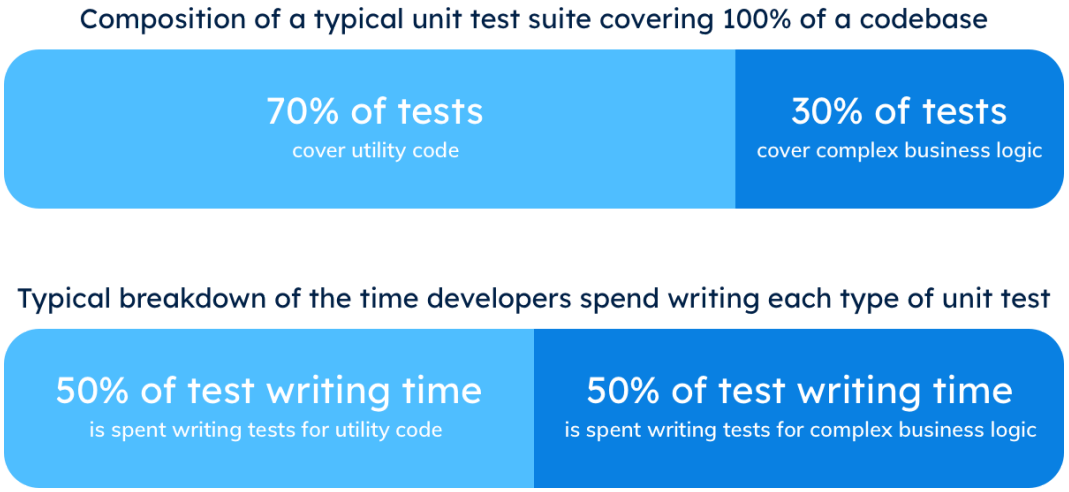
Another key advantage of AI-generated unit regression testing is the speed at which they can be created, and the subsequent time and cost savings. Though many organizations have code coverage goals of around 60-80%, quickly increasing existing code coverage by even 20% can allow developers to catch regressions that could otherwise have potentially huge impacts at later stages of the software development lifecycle. The cost of manually writing enough unit tests to increase coverage by this amount would be substantial, as demonstrated below.

Calculating savings of AI-generated unit regression tests

Findings from the **2019 Diffblue developer survey** showed that the time spent writing unit tests costs companies an average of £14,287 per developer, per year. With an average of 45 developers employed at the companies included in the study, the typical unit testing cost for a mid-size company (with at least 500 employees) is approximately £643,000 per year, plus ad hoc maintenance to keep unit tests updated after they're written.

As a thought experiment to demonstrate the time saved by unit regression tests, suppose that 30% of the possible unit tests for an application cover

complex business logic. When it comes to complex business logic, a human will often produce more accurate unit tests than AI-generated tests; these complex tests take up 50% of the time a developer spends writing unit tests.



Running Diffblue Cover will create unit regression tests that increase code coverage for utility code by an average of 35% additional coverage for the whole application. In this case, AI-generated tests for this utility code yield a 1000x increase in authoring speed. Because the tests are automatically maintained by Diffblue Cover, they require no additional time or work to keep them up-to-date.

As a result, using Diffblue Cover to create unit regression tests would save an organization on average up to 25% of their total time (and cost) of unit testing. For the organizations included in the above study, this would amount to an average savings of £160,000/year, without even accounting for the time saved by automatically maintaining the tests.

The time developers save by using unit regression tests could also be reinvested in writing more unit tests for the highly complex business logic that would benefit from a human author, further improving the quality and resilience of the company's code.

Demonstrated Savings from Diffblue Cover

In practice, Diffblue Cover validates the calculations above. For one customer, Diffblue Cover created 3,200 tests for a back-end application overnight—1,000 times faster than writing the equivalent number manually.

Time spent writing unit tests

	Manual Effort*	Diffblue Cover
Number of tests	3,211	3,211
Average time to write each	30 minutes	10 seconds
Days spent writing tests per application	268 workdays	1/3 day (run overnight)

* Manual effort assumes industry averages of 30 minutes per manual test and 6 hours productive time per day.

This saved the company over one man-year, in addition to providing more confidence in application stability when adding new code, which also improved the speed at which engineering teams could deliver business value.

Diffblue Cover also increased coverage for another module within an important backend system by 36%, and picked up on edge cases in other applications that could have led to customer-impacting incidents.

How to get started with unit regression tests

Unit regression tests are created by Diffblue Cover and can currently be made for Java code. Diffblue Cover supplements the necessary work of creating tests for teams that are already strapped for time, allowing them to spend more of their time on the most valuable and creative aspects of their work.

Diffblue Cover automatically writes tests for new code (e.g. a development branch) and because a new baseline test suite is created each time Diffblue Cover is run, the tests are always up-to-date without manually maintaining them.

Unit tests don't have to be expensive. **Sign up for a free trial of Diffblue Cover** to see how unit regression tests can efficiently improve your company's code quality, or **join one of our regular open demos** to see the tool demonstrated live. You can also **watch the latest demo** on our website.