Back to Blog

What Is Branch Coverage and What Does It Really Tell You?

What is branch coverage? That's the question we'll answer today. You'll learn what this metric is all about and how it can help you.

Published February 18, 2021 • Carlos Schults

you know that the tests you write really verify your app is working correctly? Is it possible to know for sure whether the tests you have are enough for your application?

A resource that engineers often resort to in cases like these is metrics. Tracking important metrics is a valuable way to get an objective assessment of many facets of software development, and testing is no different. In today's post, we'll tackle a metric called branch coverage. What is it? What does this metric really tell you? And what are its limitations? This is what we'll discuss in this post.

We'll start answering the "what" question by providing a quick definition of branch coverage. We'll then follow that with an explanation of how it differs from other metrics with similar names, such as code coverage and statement coverage, to name a few.

After that, we'll delve deeper into the concept of branch coverage. We'll provide examples, explaining the ways in which this metric can be useful. Finally, we'll also explain some of the important limitations of this metric. By the end of the post, you'll not only know what branch coverage is, but you'll also have a solid understanding of what this metric does and doesn't tell you.

Let's get started.

Table of Contents

code can take after a decision statement—e.g., an **if** statement—gets evaluated.

Branch coverage is an important metric in that it can help a team or organization assess whether an application has been tested to completion. A low branch coverage shows that there are scenarios in the application lacking testing. Such scenarios might contain defects that will only manifest in edge cases when the application makes it to production.

Branch coverage is an important metric in that it can help a team or organization assess whether an application has been tested to completion.

As you'll soon see, branch coverage is more nuanced than other metrics. A different metric can be at 100%, while branch coverage is lower. By only tracking the other metric, a team can have an unjustified degree of confidence in their code, and important defects might go unnoticed until they manifest in production.

Branch Coverage: How It Differs From Similar Metrics

There are many test-related metrics with similar-sounding names. Besides

Branch vs. Statement Coverage

Here's where things can get somewhat confusing. Some people struggle to understand the difference between these two metrics. After our explanation and example, you'll hopefully easily understand how they differ.

Let's start with statement coverage. This metric simply tells you the ratio of statements in an application that are currently under testing. Branch coverage, as we've seen, is about whether all branches—or paths of execution—in an application are under test.

Many people think they're equivalent, but this isn't true. While 100% of branch coverage implies 100% of statement coverage, the opposite is not true. Consider the following code excerpt:

```
if (condition) { if (condition2) { System.out.println("Hey!"); } }
```

If, in our test case, both **condition** and **condition2** are true, the string "Hey!" will be displayed on the screen. All statements will be executed. The statement coverage would be 100%. The branch coverage, on the other hand, wouldn't be 100%. Why? Easy: the scenario in which **condition2** is **false** and the string isn't displayed on the screen is not exercised through the test.

That's what we meant by saying that it's a more nuanced metric. It carries more information. Only caring about statement coverage can give teams a false sense of security when it comes to the comprehensiveness of their tests.

So, while 100% statement coverage necessarily implies 100% line coverage, the opposite isn't true. A line can contain multiple statements, but it's possible not all of them will be executed.

Finally, branch coverage differs from line coverage in a similar way to which it differs from statement coverage. That is, even if the test cases exercise all lines, that doesn't mean that it also exercises all possible logical paths.

Branch Coverage and Cyclomatic Complexity

Branch coverage is closely related to an important metric in software engineering: cyclomatic complexity.

Cyclomatic complexity, in short, is the number of possible paths of execution inside a block of code—e.g., a function. Cyclomatic complexity has many valuable use cases. You can use it to determine which portions of the code are more complex and thus more prone to defects. Cyclomatic complexity might also correlate with the difficulty of reading and maintaining a certain piece of code.

But, in our context, cyclomatic complexity is essential. It helps to determine the minimum number of test cases you need to comprehensively test a given piece of code. So striving to keep cyclomatic complexity low is a good goal to have if

- It helps you determine which logical paths got tested or not, contributing to a more comprehensive test suite.
- Also, it might help you cover areas that didn't get adequate coverage from different testing methods.
- Applying it reduces the likelihood of edge case defects making it into production.

However, it doesn't tell you everything. It won't tell you much about the quality of the tests themselves. For instance, you could achieve 100% of branch coverage even if all of your unit tests didn't contain assertions. Then, you'd be able to damage the production code, and all the tests would still pass.

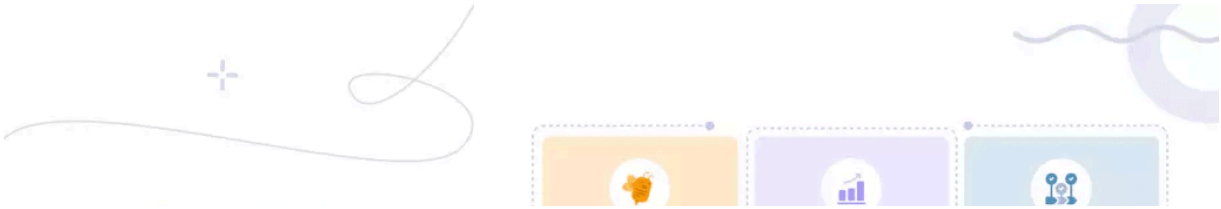
In other words, achieving a high coverage—branch or otherwise—is the bare minimum you could do. It still doesn't guarantee you have quality tests or that your code behaves as expected.

effective they are in protecting the application against defects. Even though it isn't perfect—like any other metric—branch coverage is an important way to help teams that need an objective method to assess the health of their test suites.

One thing to keep in mind is that branch coverage is probably more effective when evaluated together with other valuable metrics.

One thing to keep in mind is that branch coverage is probably more effective when evaluated together with other valuable metrics. There are other indicators that can help predict the quality of codebases in general—such as rework or code churn, for instance.

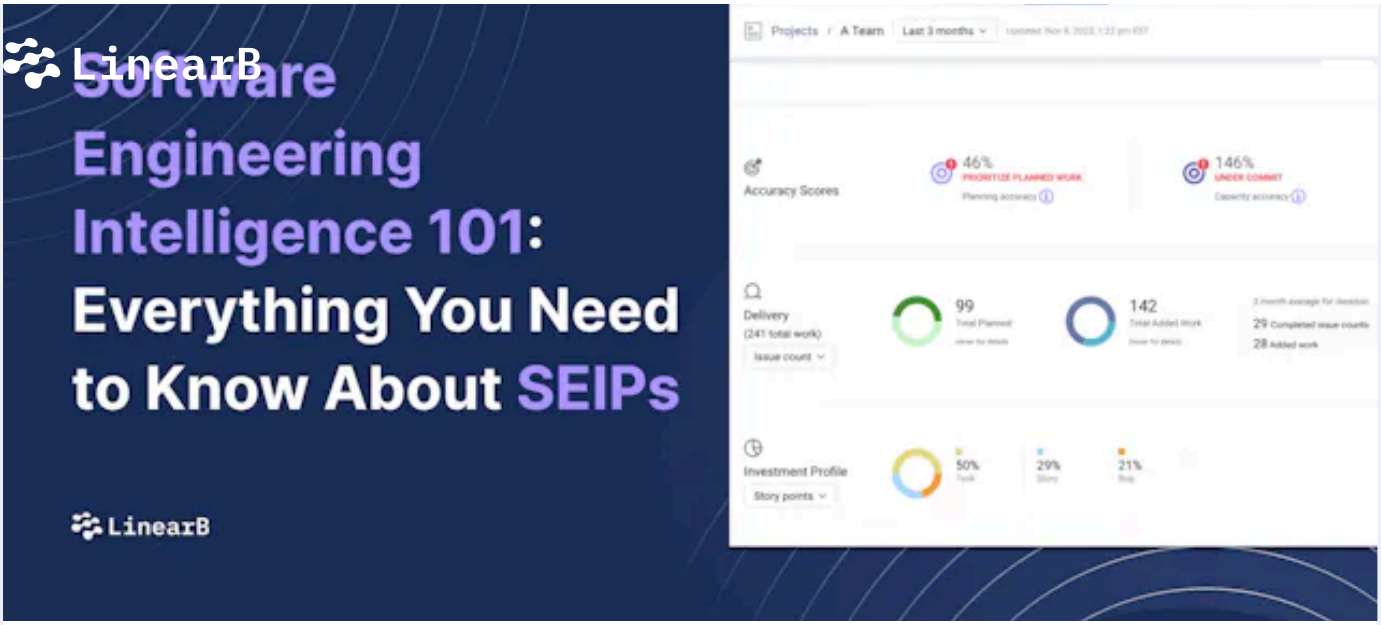
Having a comprehensive set of metrics can help smooth out the weakness of individual ones, ensuring you get a net positive result. The best way to get a single source of metrics truth and observability to give you a comprehensive look at cycle time, code churn, rework, branch coverage, and much more? Using LinearB.



MARCH 14, 2024 • NATALIE BREUER

How Rabbit Care Used LinearB to Build a Culture of Transparency While Scaling 10x

Read Now →



MARCH 7, 2024 • LINEARB

Software Engineering Intelligence 101 - Everything You Need to Know About SEIPs

Read Now →

Join our community of data-driven dev leaders