CS 240: Code Coverage Overview Transcript

[00:00:00] One of the things that can be difficult to determine when you're writing tests is when have you written enough tests.

[00:00:07] And so there is a really nice way to help you determine if you've written enough tests.

[00:00:13] And that is to use code coverage tools.

Start visual description. The professor demonstrates how to use code coverage tools to determine if enough tests have been written. End visual description.

[00:00:15] So code coverage tools are tools that can execute your code and determine what code or they, these code coverage tools can execute your tests. I mean, and then they can measure which lines of code were actually executed by the tests.

[00:00:33] So code coverage is a measure of how much code is covered or invoked by the tests that you that you invoked.

Start visual description. The professor explains that code coverage measures how much code is covered or invoked by the tests. End visual description.

[00:00:41] So um in order for this to work, these tools have to instrument the code.

[00:00:46] So that means that um they compile the code in a way where they put extra information in your code that's compiled and that extra information will report which lines of code are executed and which branches like in if and what if statements and while loops, which branches did your code go into and which did they not go into? And so, all of that information can then be reported to you.

[00:01:10] So the way this works is you compile your code with code coverage information, then you run your tests, and the tool will report how much code coverage you had.

Start visual description. The professor shows how to compile code with code coverage information and run tests to report the coverage. End visual description.

[00:01:22] Usually it will do that on a, on a class by class or file by file basis and show you individual lines that were and were not covered by the tests.

[00:01:31] So these tools can be stand alone or built into ID ES.

[00:01:35] Um It's, they're really useful to help.

[00:01:38] You know, if you've written enough tests, what they can't do is they can't determine if you um they can't determine if you've actually tested the code.

[00:01:47] So imagine for example, a case where you've written a test, but you didn't write any asserts in the test, you could write that test so it would execute some code.

[00:01:57] But if it doesn't check anything about whether the code did the right thing, then you're not actually testing anything.

[00:02:03] So, so code coverage is what we call a necessary but not sufficient condition.

[00:02:09] Just executing the code is not enough, you need to execute the code and then determine that it did the right thing.

Start visual description. The professor discusses the importance of executing code and determining that it did the right thing, emphasizing that code coverage is necessary but not sufficient. End visual description.

[00:02:15] But if you didn't execute the code, then we know you didn't have enough tests.

[00:02:19] If your tests don't execute some lines of code, then those lines of code were not, were not tested.

[00:02:25] Um But if your, if your test did execute that those lines of code, we don't know for sure that the code was tested.

[00:02:32] So code coverage isn't perfect, but it's really useful to help you uh determine if you've written enough tests.

[00:02:38] There are different measures and of code coverage and I'll talk a little bit about that in a minute and companies or teams generally will create uh a code coverage standard.

[00:02:49] So for example, a team might say you must have at least 90% statement coverage.

[00:02:54] So we'll talk about the different kinds of coverage in a minute.

[00:02:57] Um The problem with these kinds of rules or these kinds of standards is pretty much every company has them and they're useful, we need them.

[00:03:05] But the problem is there's no right number.

[00:03:08] So for example, for a lot of the code, for most code that you write, probably we should have tests that cover 100% of the code because any lines of code that you don't cover are not tested.

[00:03:18] But in some cases, we might think it's not worth uh writing the test to cover certain lines of code.

[00:03:24] So for example, your ID E can generate getters and setters. Those are lines of code.

[00:03:30] But what is the likelihood that your ID E generated a bug when it generated your getters and setters? That's pretty low.

[00:03:38] And the number of tests you can write is basically infinite.

[00:03:42] There's no limit to the number of tests that you can write.

[00:03:44] So if I'm paying you to write tests, I probably don't want to pay you to write tests that were generated by your ID E and are really unlikely to have a bug because you're never going to be able to write all the tests that might be useful.

[00:03:59] And so, um so there are cases where maybe there's some code that you shouldn't necessarily take the time to write tests for.

[00:04:09] Um Another example would be some U I code.

[00:04:11] Sometimes it can be really hard to write tests for all the U I, you know, all the different things about layout and where does a button exist might not be worth the trouble of writing tests to test every bit of our U I code.

[00:04:26] So the point I'm making here is that at first glance, you might think 100% code coverage is a good number, but there are cases where we probably want classes to have some number less than that because we don't want to pay um somebody to write those tests.

[00:04:43] Um So here are the different types of, of code coverage though we have line coverage, which is commonly used and that is the number of lines executed by your tests over the total number of lines.

Start visual description. The professor introduces different types of code coverage, including line coverage, statement coverage, and branch coverage. End visual description.

[00:04:57] So that's a percentage of the lines that were actually covered or executed by your tests.

[00:05:01] Statement coverage is the same as line coverage.

[00:05:04] If every statement is on its own line and from the code quality lectures, you'll remember that that's generally what we should do is have every statement be on its own line.

[00:05:13] So if we follow that standard, then line coverage and statement coverage are the same thing.

[00:05:18] Um If we have multiple statements on the line though, then statement coverage goes a little bit deeper than line coverage.

[00:05:24] Another form of coverage is branch coverage.

[00:05:27] So branch coverage takes into account statement coverage, but it goes deeper.

[00:05:32] So branch coverage measures um whether you considered all the conditions of different branching statements.

[00:05:39] So imagine for example, an if statement that um doesn't have an elts, so with, with code that has an if statement with no else, you can get 100%-line coverage by only checking the condition where that if statement is true.

[00:05:55] But what about the false condition? Um line coverage won't tell you if you covered the false condition in that case or not.

[00:06:02] But branch coverage would.

[00:06:03] So branch coverage measures where whether you went into all the different um possibilities of the branches of your code.

[00:06:10] So often either line or statement coverage gets used in combination with branch coverage.

[00:06:16] There are some other measures that that are generally um less used.

[00:06:20] But function coverage will tell you which functions or methods were covered by tests that can be useful to identify.

[00:06:28] So for example, maybe you have a 90% co code coverage standard and you've met that percentage, but you have some functions that aren't tested at all.

[00:06:37] Um And so function coverage would allow you to identify something like that.

[00:06:41] Um You also have coverage numbers that can tell you which of your classes were covered and which ones were not.

[00:06:48] So again, you might have high line coverage but have some number of classes that don't have any tests that execute them.

[00:06:55] And so that's where um, class or function coverage could be, could be useful.

[00:07:00] There are other measures of code coverage as well, but these are the main ones that we would care about and the ones that get used the most often are line or statement and branch coverage.

Start visual description. The professor talks about the challenges of setting a universal code coverage standard and the arbitrary nature of any percentage set as a standard. End visual description.

[00:07:13] So I'll give you just a few more thoughts on what is good coverage as I mentioned before.

[00:07:17] There's no universally accepted right answer for um what, what the number should be of your code coverage generally, the number should be high.

[00:07:28] But in certain cases, we might say that we don't want to spend the time or the effort to write tests for certain code like code that was generated by an ID E.

[00:07:38] So 11 approach is 100% branch coverage for non IDE generated code.

[00:07:43] The problem with that is the tool can't really tell what was IDE generated and what wasn't.

[00:07:48] So that becomes hard to measure.

[00:07:50] And so what we usually end up with is some percentage uh as the standard. So maybe 90% to account for those cases. Where maybe some code shouldn't be covered and then programmers are just supposed to know that um, they really should have higher coverage in cases where that makes sense.

[00:08:08] Um But just, just understand though that any percentage is arbitrary and not be, may not be appropriate for all code written by a team or an organization or a developer.