Code Coverage

Yegor Bugayenko

Lecture #15 out of 24
80 minutes

The slidedeck was presented by the author in this YouTube Video

Example, Part I

Live Code:

```
1  int fibonacci(int n) {
2    if (n <= 0) {
3      return 0;
4    }
5    if (n <= 2) {
6      return 1;
7    }
8    return fibonacci(n-1)
9      + fibonacci(n-2);
10 }
```

Test Code:

```
1  assert fibonacci(1) == 1;
2  assert fibonacci(2) == 1;
```

$$C = 7/10 = 70\%$$

Example, Part II

Live Code:

```
1  int fibonacci(int n) {
2    if (n <= 0) {
3      return 0;
4    }
5    if (n <= 2) {
6      return 1;
7    }
8    return fibonacci(n-1)
9      + fibonacci(n-2);
10 }
```

Test Code:

```
1  assert fibonacci(1) == 1;
2  assert fibonacci(2) == 1;
3
4  assert fibonacci(9) == 34;
5  assert fibonacci(10) == 55;
```

$$C = 9/10 = 90\%$$

Some Kinds of Code Coverage

- Line Coverage

- Statement Coverage

- Branch Coverage

- Condition Coverage

- Function Coverage

- Linear Code Sequence and Jump (LCSAJ) Coverage

- Modified Condition / Decision Coverage (MC/DC)

Four Kinds of Coverage

Live Code:

```
1 int foo(int x) {
2   if (x < 0) { return x; }
3   if (x > 10 || x == 0) {
4     return 42 / x;
5   } else {
6     return 1;
7   }
8 }
```
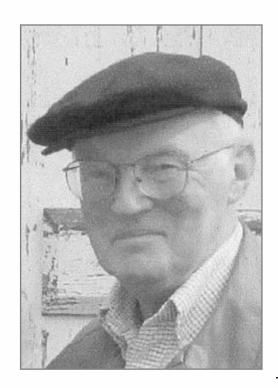
Test Code:

```
1 assert foo(1) == 1;
2 assert foo(50) == 42;
```

$$C_{\texttt{line}} = 6/6 = 100\%$$
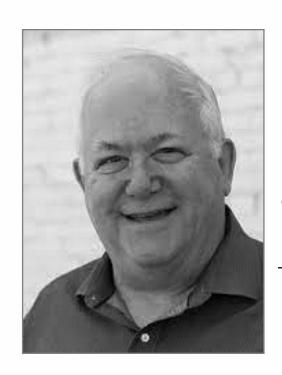
$$C_{\texttt{statement}} = 5/6 = 83\%$$

$$C_{\texttt{branch}} = 3/4 = 75\%$$

$$C_{\texttt{condition}} = 3/5 = 60\%$$

"A disciplined test control process is composed of five steps: 1) establish the intended extent of testing; 2) create a list of functional variations eligible for testing; 3) rank and subset the eligible variations so that test resources can be directed at those with the higher payoff; 4) calculate the test coverage of the test case library; and 5) verify attainment of the planned test coverage."

— William Robert Elmendorf, *Controlling the Functional Testing of an Operating System*, IEEE Transactions on Systems Science and Cybernetics, 5(4), 1969

"However, only half regularly document their test designs, only half regularly save their tests for reuse after software changes, and an extremely small five percent provide regular measurements of code coverage."

— *The Growth of Software Testing*, David Gelperin and Bill Hetzel, Communications of the ACM, 31(6), 1988

Test Practice	% Yes	% Sometimes
1 Record of defects found during testing is maintained	73	16
2 Designated person is responsible for the test process	65	13
3 Test plan describing objectives/approach is required	61	29
4 Testing is a systematic and organized activity	61	30
5 Full-time testers perform system testing	62	19
6 Testing is separated from development	60	20
7 Tests are required to be rerun when software changes	51	35
8 Tests are saved and maintained for future use	51	28
9 Test specifications and designs are documented	48	36
10 Test procedure is documented in the standards manual	45	15
11 A log of tests run is maintained	42	35
12 A record of the time spent on testing is maintained	40	30
13 Test documents are formally peer-reviewed	31	29
14 Full-time testers perform integration testing	24	24
15 The cost of testing is measured and tracked	24	19
16 Test training is provided periodically	22	26
17 Test results are formally peer reviewed	20	31
18 Users are heavily involved in test activities	8	39
19 Tests are developed before coding	8	29
20 A measurement of code coverage achieved is required	5	16

FIGURE 9.   Analysis of Industry Test Practice Usage

"We note an <u>inconsistency</u>. A high percentage of the respondents felt that the testing in their organization was a systematic and organized activity (91% answered either "yes" or "sometimes" to this practice). However, [...] an extremely small 5% provide <u>regular measurements of code coverage.</u>"

— *The Growth of Software Testing*, David Gelperin and Bill Hetzel, Communications of the ACM, 31(6), 1988

"Junky software takes more tests to achieve coverage, but it breaks under any systematic test."

— *Black Box Testing*, Boris Beizer, 1995

"Coverage numbers (like many numbers) are dangerous because they're <u>objective</u> but <u>incomplete</u>. They too often distort sensible action. Using them in isolation is as foolish as hiring based only on GPA."

— Brian Marick, *How to Misuse Code Coverage*, 1997

"I would be suspicious of anything like 100% — it would smell of someone writing tests to make the coverage numbers happy, but not thinking about what they are doing."

— Martin Fowler, *Test Coverage*, 1997

"As you get near 100 percent line coverage, that doesn't tell you the product is near release. It just tells you that the product is no longer obviously far from release according to this measure."

— Cem Kaner, James Bach, Bret Pettichord, *Lessons Learned in Software Testing: A Context-Driven Approach*, 2002

"Our results show that coverage has an <u>insignificant correlation</u> with the number of bugs that are found after the release of the software at the project level, and <u>no such correlation</u> at the file level."

— *Code Coverage and Postrelease Defects: A Large-Scale Study on Open Source Projects*, Pavneet Singh Kochhar, David Lo, Julia Lawall, Nachiappan Nagappan, IEEE Transactions on Reliability, 66(4), 2017

"Google does not enforce any code coverage thresholds across the entire codebase. Projects (or groups of projects) are free to define their own thresholds and goals. Many projects opt-into a centralized voluntary alerting system that defines five levels of code coverage thresholds."

— *Code Coverage at Google*, Goran Petrović, Marko Ivanković, René Just, Gordon Fraser, Proceedings of the 27th Joint Meeting on ESEC/FSE, 2019

Code Coverage Threshold Levels in Google

Table 2: Coverage levels and corresponding thresholds. Many projects voluntarily set these thresholds as their goal.

LEVEL	THRESHOLD
Level 1	Coverage automation disabled
Level 2	Coverage automation enabled
Level 3	Project coverage at least 60%; Changelist coverage at least 70%
Level 4	Project coverage at least 75%; Changelist coverage at least 80%
Level 5	Project coverage at least 90%; Changelist coverage at least 90%

"Code coverage <u>does not guarantee</u> that the covered lines or branches have been tested correctly, it just guarantees that they have been executed by a test. But a low code coverage number <u>does guarantee</u> that large areas of the product are going completely untested by automation on every single deployment."

— Carlos Arguelles, Marko Ivanković, and Adam Bender. Code Coverage Best Practices, 2020. URL https://testing.googleblog.com/2020/08/code-coverage-best-practices.html

Adam Bender

"Many <u>contracts</u> are specifying that a certain <u>percentage</u> of the statements or instructions must be successfully executed before the acceptance of the software by the customer."

— John B. Bowen. A Survey of Standards and Proposed Metrics for Software Quality Testing. *Computer*, 12(08):37–42, 1979

Industry Standards that Require Code Coverage

- ISO-26262: "Road Vehicles" functional safety (Switzerland)

- IEC 61508: "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems" (UK)

- DO-178C: "Software Considerations in Airborne Systems and Equipment Certification" (USA)

- IEC 62304: "Medical Device Software" (UK)

ISO-26262:

	Methods	ASIL			
		A	B	C	D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

Table 12 (Software Unit Level), ISO 26262-6

	Methods	ASIL			
		A	B	C	D
1a	Function coverage	+	+	++	++
1b	Call coverage	+	+	++	++

Table15 (Software Architectural Level), ISO 26262-6

IEC 61508:

SIL: Safety Integrity Level

Method		SIL 1	SIL 2	SIL 3	SIL 4
...	...	...	...	...	...
7a	Function Coverage	++	++	++	++
7b	Statement Coverage	+	++	++	++
7c	Branch Coverage	+	+	++	++
7d	MC/DC	+	+	+	++

Table B.2 from DIN EN 61508-3

DO-178C:

Level	Impact	Coverage Level	% of Systems	% of Software
A	Catastrophic	MC/DC, C1, C0	20-30%	40%
B	Hazardous/Severe	C1, C0	20%	30%
C	Major	C0	25%	20%
D	Minor	-	20%	10%
E	No Effect	-	10%	5%

IEC 62304:

	Methods	ASIL			
		A	B	C	D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

Table 12 (Software Unit Level), ISO 26262-6

	Methods	ASIL			
		A	B	C	D
1a	Function coverage	+	+	++	++
1b	Call coverage	+	+	++	++

Table15 (Software Architectural Level), ISO 26262-6

Codecov.io

Line Coverage

yegor256 / micromap / ⑂ master

Coverage Flags Commits Pulls Settings

micromap / src / **iterators.rs**   Uncovered ⚠ | Partial ! ⋮ | Covered

```
110        #[inline]
111        #[must_use]
112        fn into_iter(self) -> Self::IntoIter {
113            IntoIter {
114                pos: 0,
115                map: ManuallyDrop::new(self),                              ⚠
116            }
117        }
118    }
119
120    impl<K: PartialEq, V, const N: usize> Drop for IntoIter<K, V, N> {
121        fn drop(&mut self) {
122            for i in self.pos..self.map.len {
123                self.map.item_drop(i);
124            }
125        }
126    }
127
128    impl<'a, K, V> DoubleEndedIterator for Iter<'a, K, V> {
129        fn next_back(&mut self) -> Option<Self::Item> {              ⚠
130            self.iter.next_back().map(|p| {                          ⚠
131                let p = unsafe { p.assume_init_ref() };              ⚠
132                (&p.0, &p.1)                                         ⚠
133            })
134        }
135    }
```

Tarpaulin for Rust

| Code | Blame | 23 lines (23 loc) · 551 Bytes | | Raw | ⧉ | ⬇ | ✎ ▾ | <> |

```yaml
1    ---
2    name: tarpaulin
3    on:
4      push:
5        branches:
6          - master
7    jobs:
8      tarpaulin:
9        runs-on: ubuntu-22.04
10       steps:
11         - uses: actions/checkout@v4
12         - uses: actions-rs/toolchain@v1
13           with:
14             toolchain: stable
15             override: true
16         - uses: actions-rs/tarpaulin@v0.1
17           with:
18             version: '0.22.0'
19             args: '--all-features --exclude-files src/lib.rs -- --test-threads 1'
20         - uses: codecov/codecov-action@v3
21           with:
22             token: ${{ secrets.CODECOV_TOKEN }}
23             fail_ci_if_error: true
```

Code Coverage Threshold, JaCoCo Example

```
 1 <project>
 2   [...]
 3   <build>
 4     <plugins>
 5       <plugin>
 6         <groupId>org.jacoco</groupId>
 7         <artifactId>jacoco-maven-plugin</artifactId>
 8         <version>0.8.11</version>
 9         <executions>
10           <execution>
11             <id>jacoco-initialize</id>
12             <goals>
13               <goal>prepare-agent</goal>
14             </goals>
15           </execution>
16           <execution>
17             <id>jacoco-check</id>
18             <goals>
19               <goal>check</goal>
20             </goals>
21             <configuration>
22               <rules>
23                 [...]   ← Next slide
24               </rules>
25             </configuration>
26           </execution>
27           <execution>
28             <id>report</id>
29             <goals>
30               <goal>report</goal>
31             </goals>
32           </execution>
33         </executions>
34       </plugin>
35     </plugins>
36   </build>
37 </project>
```

Code Coverage Threshold, JaCoCo Rules

```
1  <rules>
2    <rule>
3      <element>BUNDLE</element>
4      <limits>
5        <limit>
6          <counter>INSTRUCTION</counter>
7          <value>COVEREDRATIO</value>
8          <minimum>0.67</minimum>
9        </limit>
10       <limit>
11         <counter>LINE</counter>
12         <value>COVEREDRATIO</value>
13         <minimum>0.84</minimum>
14       </limit>
15       <limit>
16         <counter>BRANCH</counter>
17         <value>COVEREDRATIO</value>
18         <minimum>0.47</minimum>
19       </limit>
20       <limit>
21         <counter>COMPLEXITY</counter>
22         <value>COVEREDRATIO</value>
23         <minimum>0.57</minimum>
24       </limit>
25       <limit>
26         <counter>METHOD</counter>
27         <value>COVEREDRATIO</value>
28         <minimum>0.76</minimum>
29       </limit>
30       <limit>
31         <counter>CLASS</counter>
32         <value>MISSEDCOUNT</value>
33         <maximum>2</maximum>
34       </limit>
35     </limits>
36   </rule>
37 </rules>
```

Source: https://github.com/volodya-lombrozo/jtcop

Code Coverage can be calculated by a few tools:

- JaCoCo for Java

- Istanbul for Javascript

- Gcov for C/C++

- Coverage.py for Python

- Simplecov for Ruby

- Tarpaulin for Rust

Read this:

Code Coverage Best Practices, Carlos Arguelles, Marko Ivanković, Adam Bender, Google Blog, 2020

Black Box Testing, Boris Beizer, John Wiley & Sons, 1995

References

Carlos Arguelles, Marko Ivanković, and Adam Bender. Code Coverage Best Practices, 2020. URL https://testing.googleblog.com/2020/08/code-coverage-best-practices.html.

John B. Bowen. A Survey of Standards and Proposed Metrics for Software Quality Testing. *Computer*, 12(08):37–42, 1979.