# Code Coverage

YEGOR BUGAYENKO

Lecture #15 out of 24 80 minutes

The slidedeck was presented by the author in this YouTube Video

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.

### Example, Part I

#### Live Code:

```
int fibonacci(int n) {
   if (n <= 0) {
     return 0;
   }
   if (n <= 2) {
     return 1;
   }
   return fibonacci(n-1)
   + fibonacci(n-2);
}</pre>
```

#### Test Code:

```
assert fibonacci(1) == 1; assert fibonacci(2) == 1; C = 7/10 = 70\%
```

### Example, Part II

#### Live Code:

```
int fibonacci(int n) {
   if (n <= 0) {
    return 0;
   }
   if (n <= 2) {
    return 1;
   }
   return fibonacci(n-1)
   + fibonacci(n-2);
}</pre>
```

#### Test Code:

```
assert fibonacci(1) == 1;
assert fibonacci(2) == 1;
assert fibonacci(9) == 34;
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)

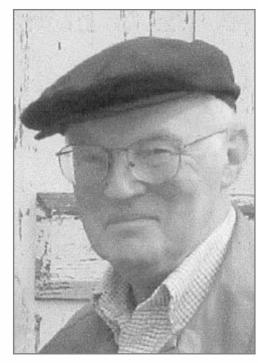
# Four Kinds of Coverage

#### Live Code:

```
int foo(int x) {
  if (x < 0) { return x; }
  if (x > 10 || x == 0) {
    return 42 / x;
  } else {
    return 1;
  }
}
```

#### Test Code:

```
| assert foo(1) == 1;
| assert foo(50) == 42;
| C_{\text{line}} = 6/6 = 100\%
| C_{\text{statement}} = 5/6 = 83\%
| C_{\text{branch}} = 3/4 = 75\%
| C_{\text{condition}} = 3/5 = 60\%
```



WILLIAM ROBERT ELMENDORF

"A disciplined test control process is composed of five steps: 1) establish the intended extent of testing; 2) create a list of functional <u>variations</u> eligible for testing; 3) rank and subset the <u>eligible</u> variations so that test resources can be directed at those with the higher payoff; 4) calculate the <u>test coverage</u> 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*, 1969. doi:10.1109/tssc.1969.300221



David Gelperin and Bill Hetzel

"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."

— D. Gelperin and B. Hetzel. The growth of software testing. *Communications of the ACM*, 1988. doi:10.1145/62959.62965

	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 <i>systematic</i> and <i>organized</i> 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
	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
	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
	A measurement of code coverage achieved is required	5	16

FIGURE 9. Analysis of Industry Test Practice Usage

"We note an inconsistency. 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 regular measurements of code coverage."

<sup>-</sup> The Growth of Software Testing, David Gelperin and Bill Hetzel, Communications of the ACM, 31(6), 1988



Boris Beizer

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

— Boris Beizer. *Black Box Testing*. ACM Digital Library, 1995. doi:10.5555/202699



BRIAN MARICK

"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."

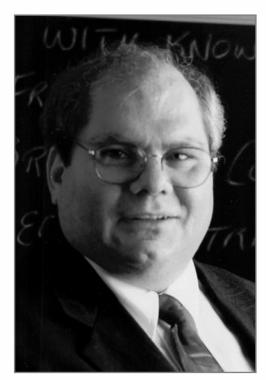
— Brian Marick. How to Misuse Code Coverage. In *Proceedings of the 16th Interational Conference on Testing Computer Software*, 1997. URL http://www.exampler.com/testing-com/writings/coverage.pdf



MARTIN FOWLER

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

— Martin Fowler. Continuous Integration, 1997. URL https://martinfowler.com/bliki/TestCoverage.html



 $\frac{\mathsf{Cem}\;\mathsf{Kaner},\,\mathsf{James}\;\mathsf{Bach},}{\mathsf{Bret}\;\mathsf{Pettichord}}$ 

"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 <u>obviously far</u> from release according to this measure."

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



Pavneet Singh Kochhar, David Lo, Julia Lawall, Nachiappan Nagappan

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

— Pavneet Singh Kochhar, David Lo, Julia Lawall, and Nachiappan Nagappan. Code Coverage and Postrelease Defects: A Large-Scale Study on Open Source Projects. *IEEE Transactions on Reliability*, 2017. doi:10.1109/tr.2017.2727062



Goran Petrović, Marko Ivanković, René Just, Gordon Fraser

"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."

— Goran Petrović, Marko Ivanković, René Just, and Gordon Fraser. Code coverage at Google. In *Proceedings of the 27th Joint Meeting on ESEC/FSE*, 2019. doi:10.1145/3338906.3340459. URL

https://dl.acm.org/doi/abs/10.1145/3338906.3340459

# Code Coverage Threshold Levels in Google

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

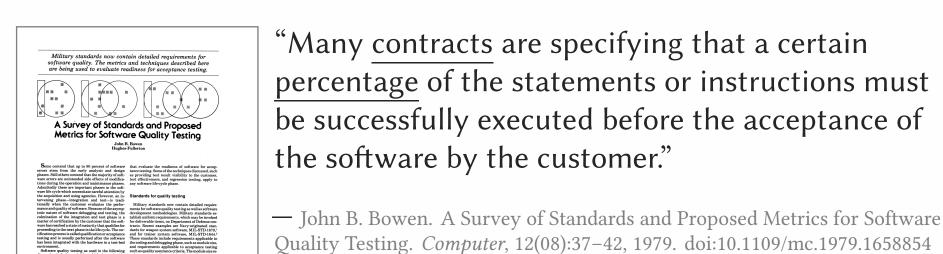
LEVEL	Threshold
Level 1	Coverage automation disabled
Level 2	Coverage automation enabled
Level 3	Project coverage at least 60%; Changelist coverage at least 70%
Level 4	Project coverage at least 75%; Changelist coverage at least 80%
Level 5	Project coverage at least 90%; Changelist coverage at least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



Adam Bender

"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. https://testing.googleblog.com/2020/08/code-coverage-best-practices.html, 2020. [Online; accessed 15-03-2024]



## 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	В	C	D		
la	Statement coverage	++	++	+	+		
1b	Branch coverage	+	++	++	++		
lc	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++		

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

	Methods	ASIL			
		A	В	C	D
la	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	<b>Coverage Level</b>	% of Systems	% of Software
A	Catastrophic	MC/DC, C1, C0	20-30%	40%
В	Hazardous/Severe	C1, C0	20%	30%
С	Major 🔓	со	25%	20%
D	Minor	-	20%	10%
E	No Effect	-	10%	5%

#### IEC 62304:

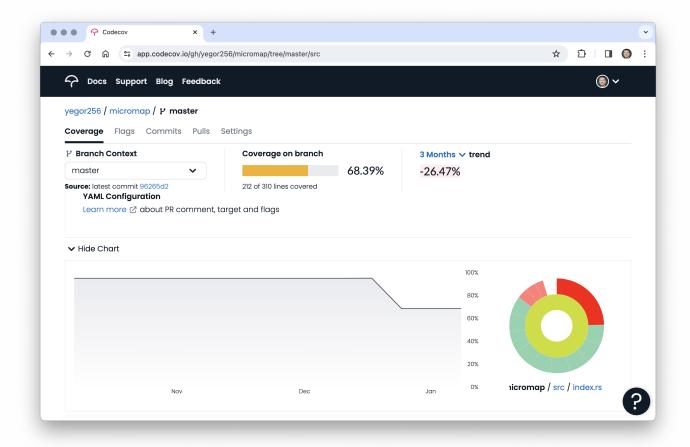
	Methods		ASIL				
		A	В	C	D		
la	Statement coverage	++	++	+	+		
1b	Branch coverage	+	++	++	++		
lc	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++		

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

Methods ASIL		IL	_		
		A	В	C	D
la	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
                                                                              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
  impl<K: PartialEq, V, const N: usize> Drop for IntoIter<K, V, N> {
          fn drop(&mut self) {
               for i in self.pos..self.map.len {
  122
  123
                  self.map.item_drop(i);
  124
  125
  126 }
  127
  impl<'a, K, V> DoubleEndedIterator for Iter<'a, K, V> {
          fn next_back(&mut self) -> Option<Self::Item> {
              self.iter.next_back().map(|p| {
  130
  131
                  let p = unsafe { p.assume_init_ref() };
  132
                  (&p.0, &p.1)
  133
              })
  134
  135 }
```

# Tarpaulin for Rust

```
Raw [□ 🕹 🖉 🕶 🐼
Code
        Blame 23 lines (23 loc) · 551 Bytes
         name: tarpaulin
           push:
             branches:
               - master
         jobs:
           tarpaulin:
             runs-on: ubuntu-22.04
  10
             steps:
              - uses: actions/checkout@v4
  11
  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'
                  args: '--all-features --exclude-files src/lib.rs -- --test-threads 1'
  19
  20
               - uses: codecov/codecov-action@v3
  21
                 with:
  22
                  token: ${{ secrets.CODECOV_TOKEN }}
  23
                  fail_ci_if_error: true
```

# Code Coverage Threshold, JaCoCo Example

```
</goals>
                                                               20
                                                               21
                                                                              <configuration>
     [\ldots]
     <build>
                                                               22
                                                                               <rules>
                                                               23
                                                                                 \lceil \ldots \rceil \leftarrow \text{Next slide}
       <plugins>
                                                               24
        <plugin>
                                                                               </rules>
          <groupId>org.jacoco</groupId>
                                                               25
                                                                             </configuration>
                                                                            </execution>
          <artifactId>jacoco-maven-plugin</artifactId>
                                                               26
          <version>0.8.11
                                                               27
                                                                            <execution>
                                                               28
                                                                              <id>report</id>
          <executions>
                                                               29
                                                                              <goals>
10
            <execution>
              <id>jacoco-initialize</id>
                                                                               <goal>report</goal>
                                                               30
11
12
              <goals>
                                                               31
                                                                             </goals>
                <goal>prepare-agent
                                                                            </execution>
13
                                                               32
              </goals>
                                                               33
                                                                          </executions>
14
                                                               34
                                                                        </plugin>
15
            </execution>
                                                               35
16
            <execution>
                                                                      </plugins>
17
              <id>jacoco-check</id>
                                                                    </build>
              <goals>
                                                               37 </project>
18
19
                <goal>check</goal>
```

# Code Coverage Threshold, JaCoCo Rules

```
<rules>
                                                              20
                                                                       imit>
     <rule>
                                                                        <counter>COMPLEXITY</counter>
                                                             21
                                                                        <value>COVEREDRATIO</value>
       <element>BUNDLE</element>
                                                              23
                                                                        <minimum>0.57</minimum>
      <limits>
        imit>
                                                              24
                                                                      </limit>
          <counter>INSTRUCTION</counter>
                                                                       imit>
          <value>COVEREDRATIO</value>
                                                              26
                                                                        <counter>METHOD</counter>
                                                                        <value>COVEREDRATIO</value>
          <minimum>0.67</minimum>
                                                              27
                                                                        <minimum>0.76</minimum>
        </limit>
                                                              28
        imit>
                                                              29
                                                                      </limit>
10
          <counter>LINE</counter>
                                                              30
                                                                       imit>
11
          <value>COVEREDRATIO</value>
                                                                        <counter>CLASS</counter>
12
                                                             31
          <minimum>0.84</minimum>
                                                                        <value>MISSEDCOUNT</value>
13
        </limit>
                                                              33
                                                                        <maximum>2</maximum>
14
15
        imit>
                                                             34
                                                                      </limit>
          <counter>BRANCH</counter>
16
                                                              35
                                                                    </limits>
          <value>COVEREDRATIO</value>
                                                                   </rule>
                                                             37 </rules>
          <minimum>0.47</minimum>
18
19
        </limit>
```

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

### Code Coverage can be calculated by a few tools:

- JaCoCo for Java
- Istanbul for Javascript
- $\bullet$  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.
  - https://testing.googleblog.com/2020/08/code-coverage-best-practices.html, 2020. [Online; accessed 15-03-2024].
- Boris Beizer. *Black Box Testing*. ACM Digital Library, 1995. doi:10.5555/202699.
- John B. Bowen. A Survey of Standards and Proposed Metrics for Software Quality Testing. *Computer*, 12(08):37–42, 1979. doi:10.1109/mc.1979.1658854.
- William Robert Elmendorf. Controlling the Functional Testing of an Operating System. *IEEE Transactions on Systems Science and Cybernetics*, 1969. doi:10.1109/tssc.1969.300221.
- Martin Fowler. Continuous Integration, 1997. URL https://martinfowler.com/bliki/ TestCoverage.html.
- D. Gelperin and B. Hetzel. The growth of software

- testing. *Communications of the ACM*, 1988. doi:10.1145/62959.62965.
- Cem Kaner, James Bach, and Bret Pettichord. *Lessons Learned in Software Testing: A Context-Driven Approach*. Wiley, 2002.
- Pavneet Singh Kochhar, David Lo, Julia Lawall, and Nachiappan Nagappan. Code Coverage and Postrelease Defects: A Large-Scale Study on Open Source Projects. *IEEE Transactions on Reliability*, 2017. doi:10.1109/tr.2017.2727062.
- Brian Marick. How to Misuse Code Coverage. In Proceedings of the 16th Interational Conference on Testing Computer Software, 1997. URL http://www.exampler.com/testing-com/ writings/coverage.pdf.
- Goran Petrović, Marko Ivanković, René Just, and Gordon Fraser. Code coverage at Google. In *Proceedings of the 27th Joint Meeting on ESEC/FSE*, 2019. doi:10.1145/3338906.3340459. URL https://dl.acm.org/doi/abs/10.1145/3338906.3340459.