CCIS كلية علوم الحاسب والمعلومات COLLEGE OF COMPUTER & INFORMATION SCIENCES

جامعة الأمير سلطان PRINCE SULTAN UNIVERSITY

Code Coverage

Outline

- Introduction
- The Benefits of Code Coverage
- Some Best Practices
- Google Code Coverage Best Practices

Introduction

- Code coverage identifies the percentage of your codebase that was exercised by tests.
- This percentage, which is displayed in a code coverage report, is formed through the branches, statements, functions and lines of code covered by a test suite.
- When working on a product delivery team, developers are often required to ensure that the code they write is supported by associated unit/integration/e2e tests.
- Some teams even go as far as requiring each incoming branch meet a certain code coverage threshold trusting that the high percentage eliminates bugs from the production codebase.

Introduction

- The coverage report can be helpful in identifying untested code, *but targeting a percentage of coverage does not equal quality-tested code.*

- Code coverage itself is not bad, but it provides a false sense of security.

- Some General Remarks:
  - Developers do just enough
  - Coverage does not ensure quality
  - High code coverage should not remove the need for a code review

The Benefits of Code Coverage

- The whole idea of a coverage report is that it can help developers identify what portions of their codebase is covered by unit tests. And visa versa.
- The first thing you should do when rolling onto an existing project is run a coverage report to determine where the focus should be when writing tests.
- Since coverage reports include the coverage percentage for branches, statements and functions, it's easy to pinpoint which files/components are lacking associated tests.

Code Coverage

- Some clients feel a necessity for a high coverage percentage for their product, and that's okay.
- As developers and consultants, our job is to help the client understand what they need in an application solution.
- While a high coverage percentage isn't the most important when considering all of the ingredients of a successful project, it's certainly something we can consider.

Code Coverage

- If a team is utilizing CI/CD pipelines, a flow can automatically fail if the branch has low code coverage, keeping the production branch more pure.
- If a team follows Test Driven Development, coverage percentages should be within 80%-90% at all times automatically.
- Some TDD enthusiasts believe that 100% coverage should always be attainable — but because of weird quirks of a language or framework, that may not be the case.

Summary

- A high coverage percentage does not equal quality-tested code.
- A target coverage threshold should not take the place of code reviews.
- Instead of putting most of the focus on a high percentage of code coverage, developers should understand (and experience) the incredible value of well-formed tests.
- TDD, by definition, should allow developers to attain high coverage percentages.

Some Best Practices

- Your initial set of tests should be derived from the specification of the program.
- What does it need to do? Are there corner cases based on the requirements? Are there special cases that the program needs to handle? And etc.
- Once you engineered all the tests you could see, you now may want to use coverage as a way “to see if you forgot something”.

Some Best Practices

- Imagine the coverage report telling you that you missed a branch.
- When that happens, the question you should ask yourself is: *"why didn't I see this when I was creating the tests?"*.
- Maybe you simply forgot about it. Good, that's how coverage shows its value.
- Maybe you did have a reason not to test that case. That's good again, as the coverage made you reflect about it.

Google Code Coverage Best Practices

Google Code Coverage Best Practices

- One of the areas that experienced software engineers consistently advocated for is the use of code coverage data to assess risk and identify gaps in testing.
- However, the value of code coverage is a highly debated subject with strong opinions, and a surprisingly polarizing topic.
- Every time code coverage is mentioned in any large group of people, seemingly endless arguments ensue.

Google Code Coverage Best Practices

- These tend to lead the conversation away from any productive progress, as people securely bunker in their respective camps.
- The purpose of these best practices is to give you tools to steer people on all ends of the spectrum to find common ground so that you can move forward and use coverage information pragmatically.
- These best practices help the domain of code coverage to work effectively with code health.

Google Code Coverage Best Practices

- **Code coverage provides significant benefits to the developer workflow.**
  - It is not a perfect measure of test quality, but it does offer a reasonable, objective, industry standard metric with actionable data.
  - It does not require significant human interaction, it applies universally to all products, and there are ample tools available in the industry for most languages.
  - You must treat it with the understanding that it's indirect metric that compresses a lot of information into a single number so it should not be your only source of truth.
  - Instead, use it in conjunction with other techniques to create a more holistic assessment of your testing efforts.

Google Code Coverage Best Practices

- **It is an open research question whether code coverage alone reduces defects,**
  - but experience shows that efforts in increasing code coverage can often lead to culture changes in engineering excellence that in the long run reduce defects.
  - For example, teams that give code coverage priority tend to treat testing as a first class citizen, and tend to bake stronger testability into their product design, so that they can achieve their testing goals with less effort.
  - All this in turn leads to writing higher quality code to begin with (more modular, cleaner contracts in their APIs, more manageable code reviews, etc.).
  - They also start caring more about their overall health, and engineering and operational excellence.

Google Code Coverage Best Practices

- **A high code coverage percentage does not guarantee high quality in the test coverage.**
  - Focusing on getting the number as close as possible to 100% leads to a false sense of security.
  - It could also be wasteful, burning machine cycles and creating technical debt from low-value tests that now need to be maintained.
  - Bad code being pushed to production due to missing tests could happen either because
    - (a) your tests did not cover a specific path of code, a test gap that is easy to identify with code coverage analysis, or
    - (b) because your tests did not cover a specific edge case in an area that did have code coverage, which is difficult or impossible to catch with code coverage analysis.
  - Code coverage does not guarantee that the covered lines or branches have been tested *correctly*, it just guarantees that they have been executed by a test. Be mindful of copy/pasting tests just for the sake of increasing coverage, or adding tests with little actual value, to comply with the number.
  - A better technique to assess whether you're adequately exercising the lines your tests cover, and adequately asserting on failures, is mutation testing.

Google Code Coverage Best Practices

- **But a low code coverage number does guarantee that large areas of the product are going completely untested**
  - by automation on every single deployment.
  - This increases our risk of pushing bad code to production, so it should receive attention.
  - *In fact a lot of the value of code coverage data is to highlight not what's covered, but what's not covered.*

Google Code Coverage Best Practices

- **There is no "ideal code coverage number" that universally applies to all products.**
  - The level of testing you want/need for a set of code should be a function of
    - (a) business impact/criticality of the code;
    - (b) how often you will need to touch/change the code;
    - (c) how much longer you expect the code to live, its complexity, and domain variables.
  - We cannot mandate every single team should have x% code coverage; this is a business decision best made by the owners of the product with domain-specific knowledge.
  - Any mandate to reach x% code coverage should be accompanied by infrastructure investments to make testing easy, such as integrating tools into the developer workflow.
  - Be mindful that engineers may start treating your target like a checkbox and avoid increasing coverage beyond the target, even if doing so would be prudent.

Google Code Coverage Best Practices

- **In general code coverage of a lot of products is below the bar; we should aim at significantly improving code coverage across the board.**
  - Although there is no "ideal code coverage number," at Google we offer the general guidelines of
    - 60% as "acceptable"
    - 75% as "commendable" and
    - 90% as "exemplary."
  - However we like to stay away from broad top-down mandates and encourage every team to select the value that makes sense for their business needs.

Google Code Coverage Best Practices

- **We should not be obsessing on how to get from 90% code coverage to 95%.**
  - The gains of increasing code coverage beyond a certain point are logarithmic.
  - But we should be taking concrete steps to get from 30% to 70% and always making sure new code meets our desired threshold.

Google Code Coverage Best Practices

- **More important than the percentage of lines covered is human judgment over the actual lines of code (and behaviors) that aren't being covered**
  - (analyzing the gaps in testing) and whether this risk is acceptable or not. What's not covered is more meaningful than what is covered.
  - Pragmatic discussions over specific lines of code not covered that take place during the code review process are more valuable than over-indexing on an arbitrary target number.
  - We have found out that embedding code coverage into your code review process makes code reviews faster and easier.
  - Not all code is equally important, for example testing debug log lines is often not as important, so when developers can see not just the coverage number, but each covered line highlighted as part of the code review, they will make sure that the most important code is covered.

Google Code Coverage Best Practices

- **Just because your product has low code coverage doesn't mean you can't take concrete, incremental steps to improve it over time.**
  - Inheriting a legacy system with poor testing and poor testability can be daunting, and you may not feel empowered to turn it around, or even know where to start.
  - But at the very least, you can adopt the 'boy-scout rule' (leave the campground cleaner than you found it). Over time, and incrementally, you will get to a healthy location.

Google Code Coverage Best Practices

- **Make sure that frequently changing code is covered.**
  - While project wide goals above 90% are most likely not worth it, per-commit coverage goals of 99% are reasonable, and 90% is a good lower threshold.
  - We need to ensure that our tests are not getting worse over time.

Google Code Coverage Best Practices

- **Unit test code coverage is only a piece of the puzzle.**
  - Integration/System test code coverage is important too.
  - And the aggregate view of the coverage of all sources in your Pipeline (unit and integration) is paramount, as it gives you the bigger picture of how much of your code is not exercised by your test automation as it makes its way in your pipeline to a production environment.
  - One thing you should be aware of is while unit tests have high correlation between executed and evaluated code, some of the coverage from integration tests and end-to-end tests is incidental and not deliberate.
  - But incorporating code coverage from integration tests can help you avoid situations where you have a false sense of security that even though you're not covering code in your unit tests, you think you're covering it in your integration tests.

Google Code Coverage Best Practices

- **We should gate deployments that do not meet our code coverage standards.**
  - Teams should debate and decide which gating mechanism makes sense to them.
  - You should however be careful that it doesn't turn into being treated as a checkbox that is required to be filled, as it can backfire (pressure to 'hit the metric' almost never yields the desired outcome).
  - There are many mechanisms available: gate on coverage for all code vs gate on coverage to new code only; gate on a specific hard-coded code coverage number vs gate on delta from prior version, specific parts of the code to ignore or focus on.
  - And then, commit to upholding these as a team. Drops in code coverage violating the gate should prevent the code from being checked in and reaching production.


QUESTIONS
Q&A
ANSWERS

References

- https://www.effective-software-testing.com/why-do-developers-hate-code-coverage
- https://testing.googleblog.com/2020/08/code-coverage-best-practices.html