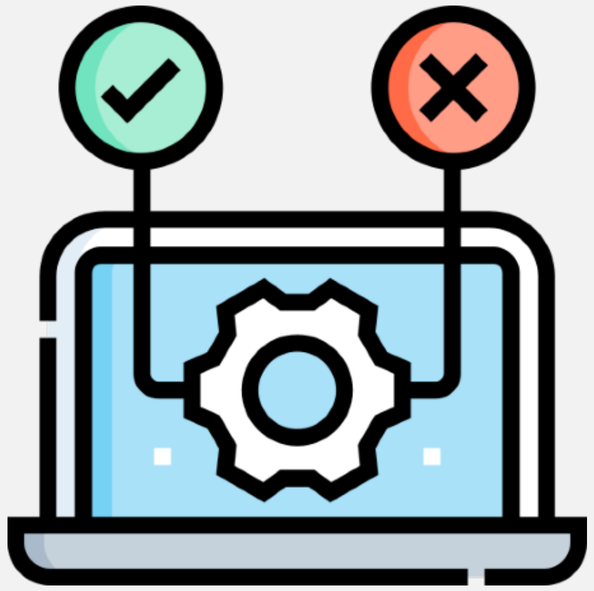
CEN 5016: Software Engineering

Fall 2025

University of Central Florida

Dr. Kevin Moran

Week 4 - Class I: Software Testing



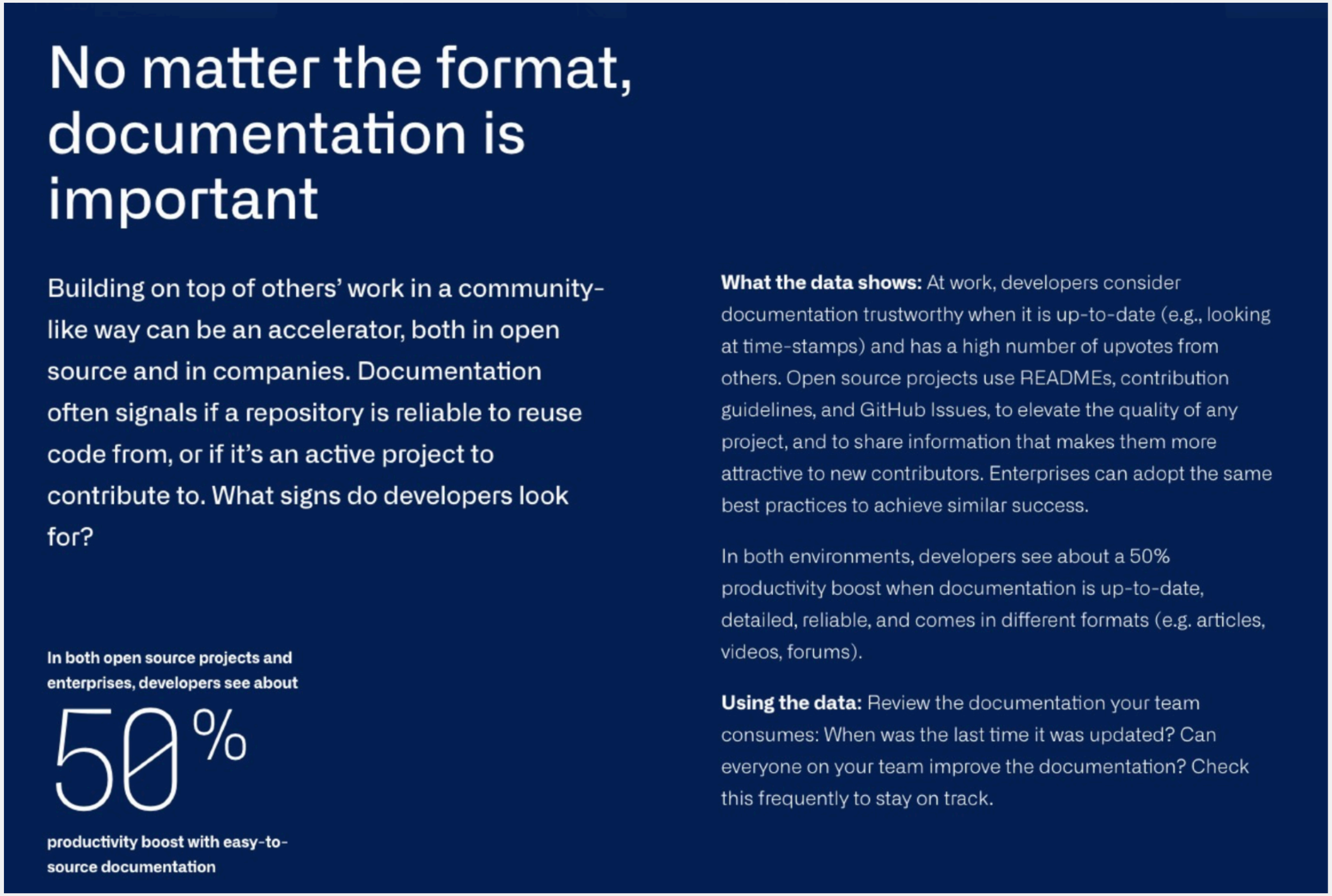
Administrivia

- *Team-forming due by Thursday!*
  - If you are not on a team, let me know and I can try to help
- *Assignment 2 Posted*
  - Getting familiar with FakeFlix, the subject of our SDE project
  - Both parts of the Assignment due Sept 12th!
  - I have posted resources related to Javascript and React from my past web dev courses to assist.
  - Get started today!
- *SDE Project Checkpoint 1 will be posted by Thursday.*

Software Teams & Communication

Knowledge Sharing

Importance of Documentation



No matter the format, documentation is important

Building on top of others' work in a community-like way can be an accelerator, both in open source and in companies. Documentation often signals if a repository is reliable to reuse code from, or if it's an active project to contribute to. What signs do developers look for?

In both open source projects and enterprises, developers see about

50%

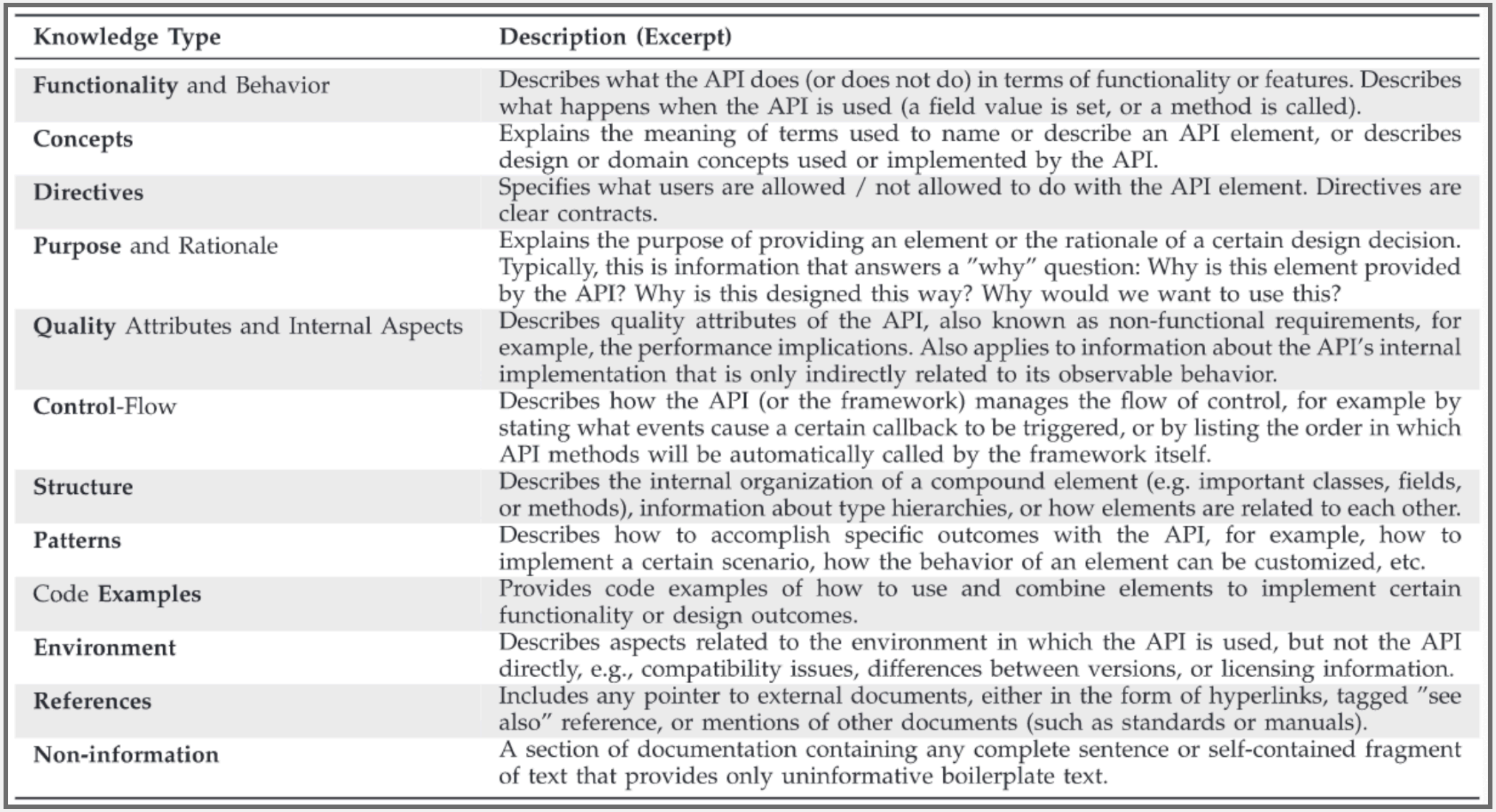
productivity boost with easy-to-source documentation

What the data shows: At work, developers consider documentation trustworthy when it is up-to-date (e.g., looking at time-stamps) and has a high number of upvotes from others. Open source projects use READMEs, contribution guidelines, and GitHub Issues, to elevate the quality of any project, and to share information that makes them more attractive to new contributors. Enterprises can adopt the same best practices to achieve similar success.

In both environments, developers see about a 50% productivity boost when documentation is up-to-date, detailed, reliable, and comes in different formats (e.g. articles, videos, forums).

Using the data: Review the documentation your team consumes: When was the last time it was updated? Can everyone on your team improve the documentation? Check this frequently to stay on track.

Types of Documentation



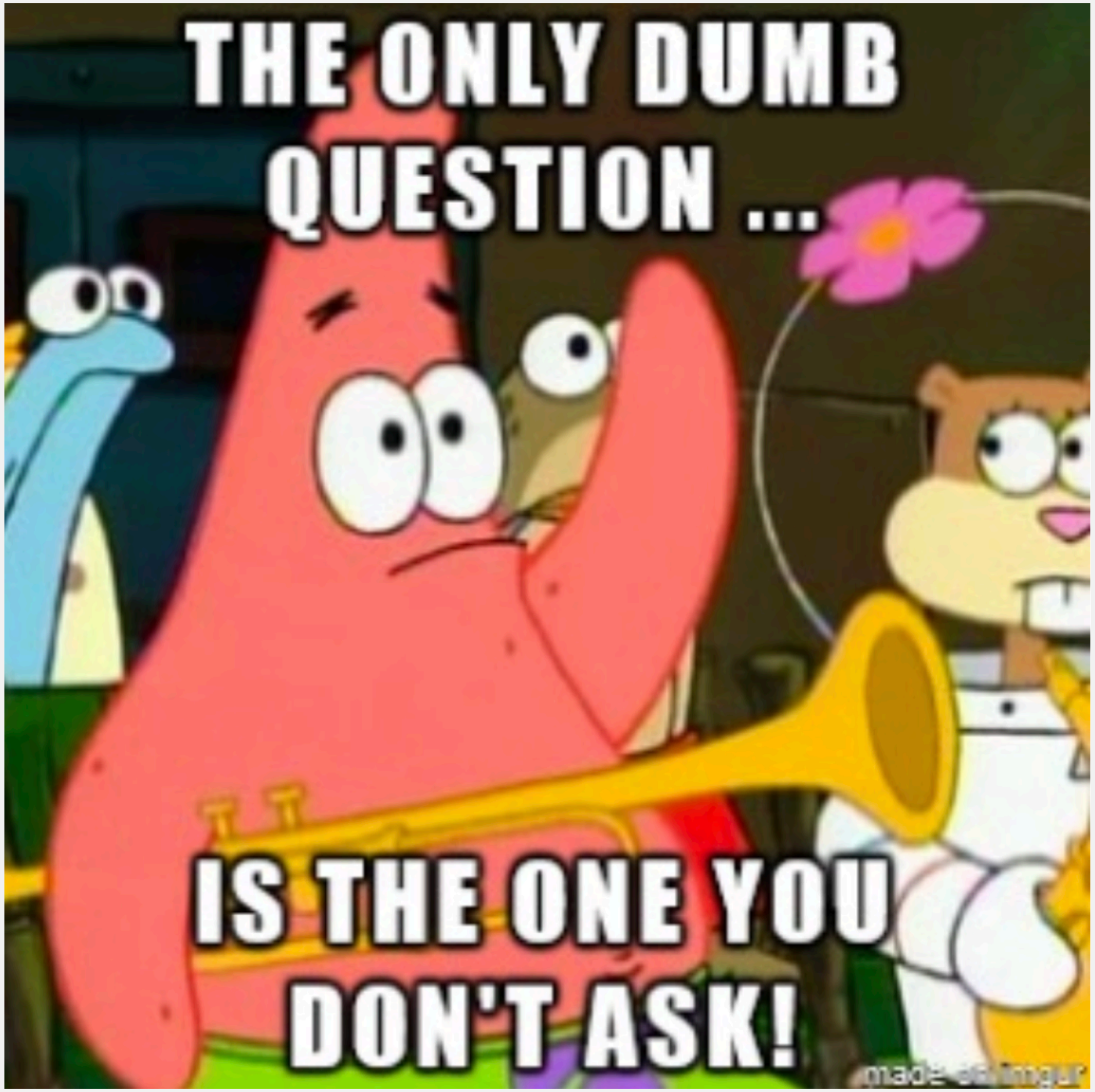
Knowledge Type	Description (Excerpt)
Functionality and Behavior	Describes what the API does (or does not do) in terms of functionality or features. Describes what happens when the API is used (a field value is set, or a method is called).
Concepts	Explains the meaning of terms used to name or describe an API element, or describes design or domain concepts used or implemented by the API.
Directives	Specifies what users are allowed / not allowed to do with the API element. Directives are clear contracts.
Purpose and Rationale	Explains the purpose of providing an element or the rationale of a certain design decision. Typically, this is information that answers a ”why” question: Why is this element provided by the API? Why is this designed this way? Why would we want to use this?
Quality Attributes and Internal Aspects	Describes quality attributes of the API, also known as non-functional requirements, for example, the performance implications. Also applies to information about the API's internal implementation that is only indirectly related to its observable behavior.
Control-Flow	Describes how the API (or the framework) manages the flow of control, for example by stating what events cause a certain callback to be triggered, or by listing the order in which API methods will be automatically called by the framework itself.
Structure	Describes the internal organization of a compound element (e.g. important classes, fields, or methods), information about type hierarchies, or how elements are related to each other.
Patterns	Describes how to accomplish specific outcomes with the API, for example, how to implement a certain scenario, how the behavior of an element can be customized, etc.
Code **Examples**	Provides code examples of how to use and combine elements to implement certain functionality or design outcomes.
Environment	Describes aspects related to the environment in which the API is used, but not the API directly, e.g., compatibility issues, differences between versions, or licensing information.
References	Includes any pointer to external documents, either in the form of hyperlinks, tagged ”see also” reference, or mentions of other documents (such as standards or manuals).
Non-information	A section of documentation containing any complete sentence or self-contained fragment of text that provides only uninformative boilerplate text.

Maalej, W., & Robillard, M. P. (2013). Patterns of knowledge in API reference documentation. IEEE Transactions on Software Engineering, 39(9), 1264-1282.

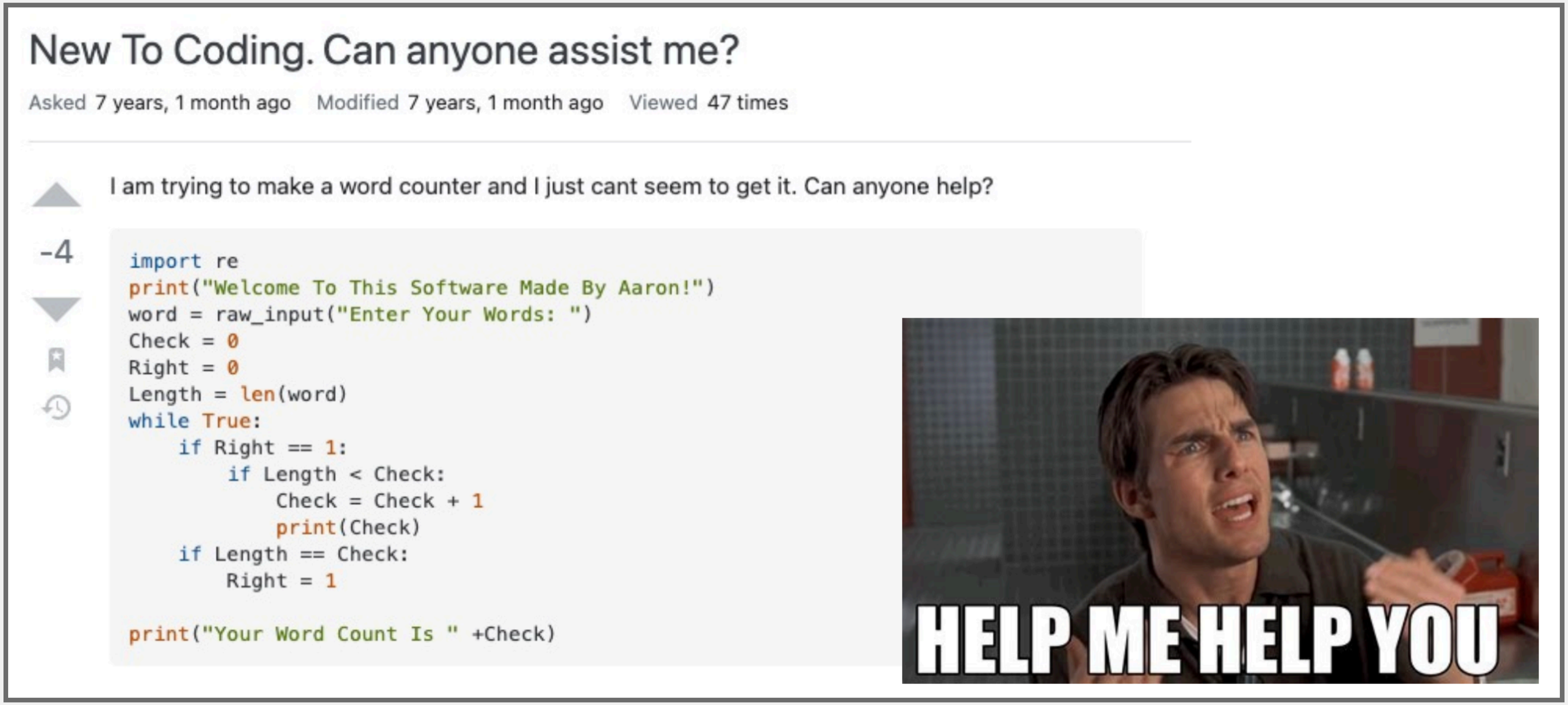
Know Your Audience

- Internal document for your team (e.g., meeting note)
- Documentation for project contributors
- Documentation for non-developer collaborators (e.g., UX researchers)
- Documentation for developer users
- Documentation for clients with no software knowldge
- User manual for end users

Importance of Asking Questions



How to Ask Questions



New To Coding. Can anyone assist me?

Asked 7 years, 1 month ago Modified 7 years, 1 month ago Viewed 47 times

-4

I am trying to make a word counter and I just cant seem to get it. Can anyone help?

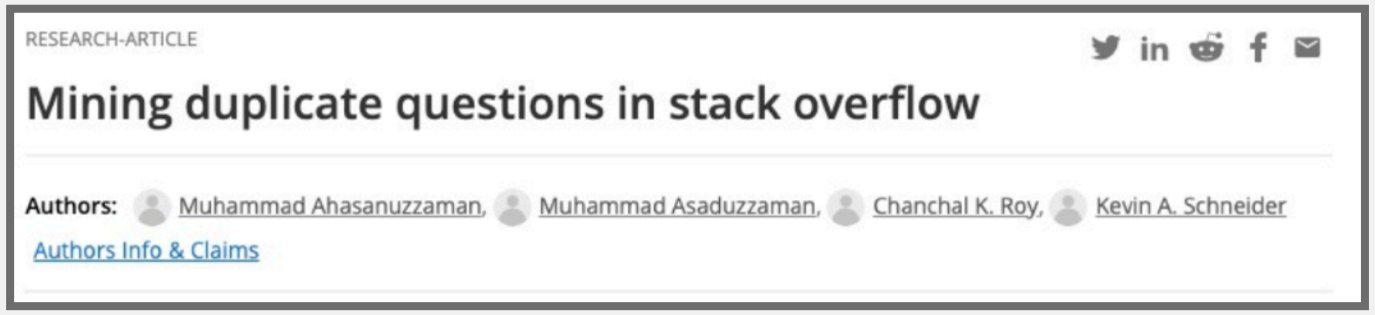
```
import re
print("Welcome To This Software Made By Aaron!")
word = raw_input("Enter Your Words: ")
Check = 0
Right = 0
Length = len(word)
while True:
    if Right == 1:
        if Length < Check:
            Check = Check + 1
            print(Check)
    if Length == Check:
        Right = 1

print("Your Word Count Is " +Check)
```

Make it Easy for People to Help You

- I am trying to ___, so that I can ___. I am running into ___.
  I have looked at ___ and tried ___.
- + I'm using this tech stack: ___.
- + I'm getting this error/result: ___.
- + I think the problem could be ___.

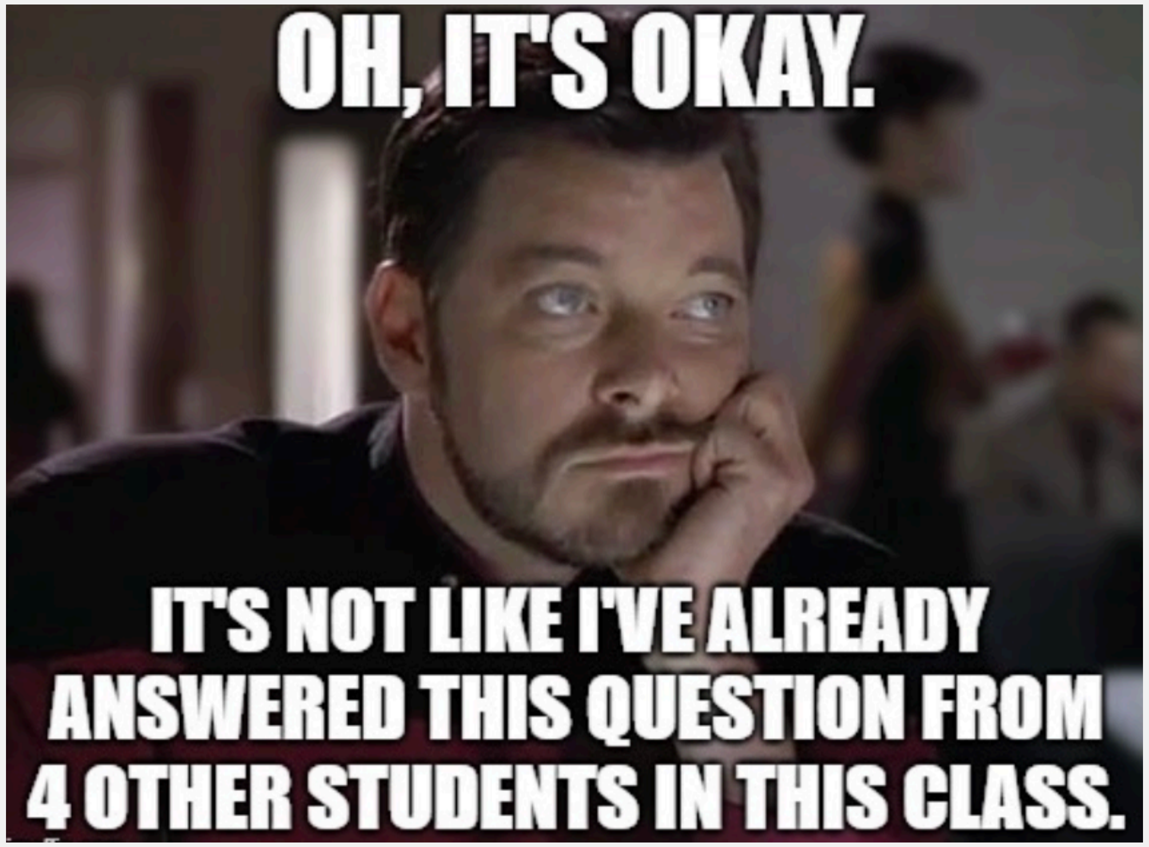
Avoid Duplication

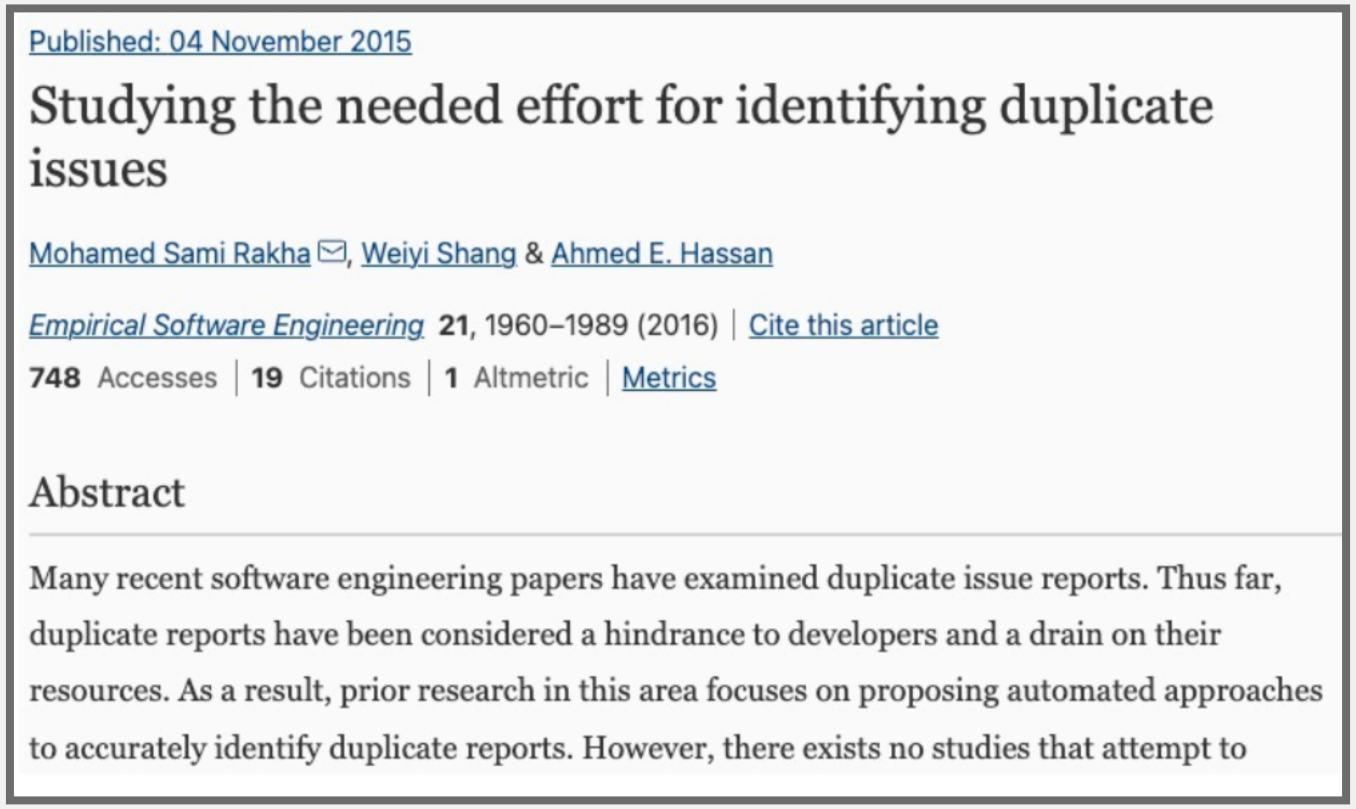

RESEARCH-ARTICLE

Mining duplicate questions in stack overflow

Authors: Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, Kevin A. Schneider

Authors Info & Claims


OH, IT'S OKAY.
IT'S NOT LIKE I'VE ALREADY ANSWERED THIS QUESTION FROM 4 OTHER STUDENTS IN THIS CLASS.


Published: 04 November 2015

Studying the needed effort for identifying duplicate issues

Mohamed Sami Rakha, Weiyi Shang & Ahmed E. Hassan

Empirical Software Engineering **21**, 1960–1989 (2016) | Cite this article

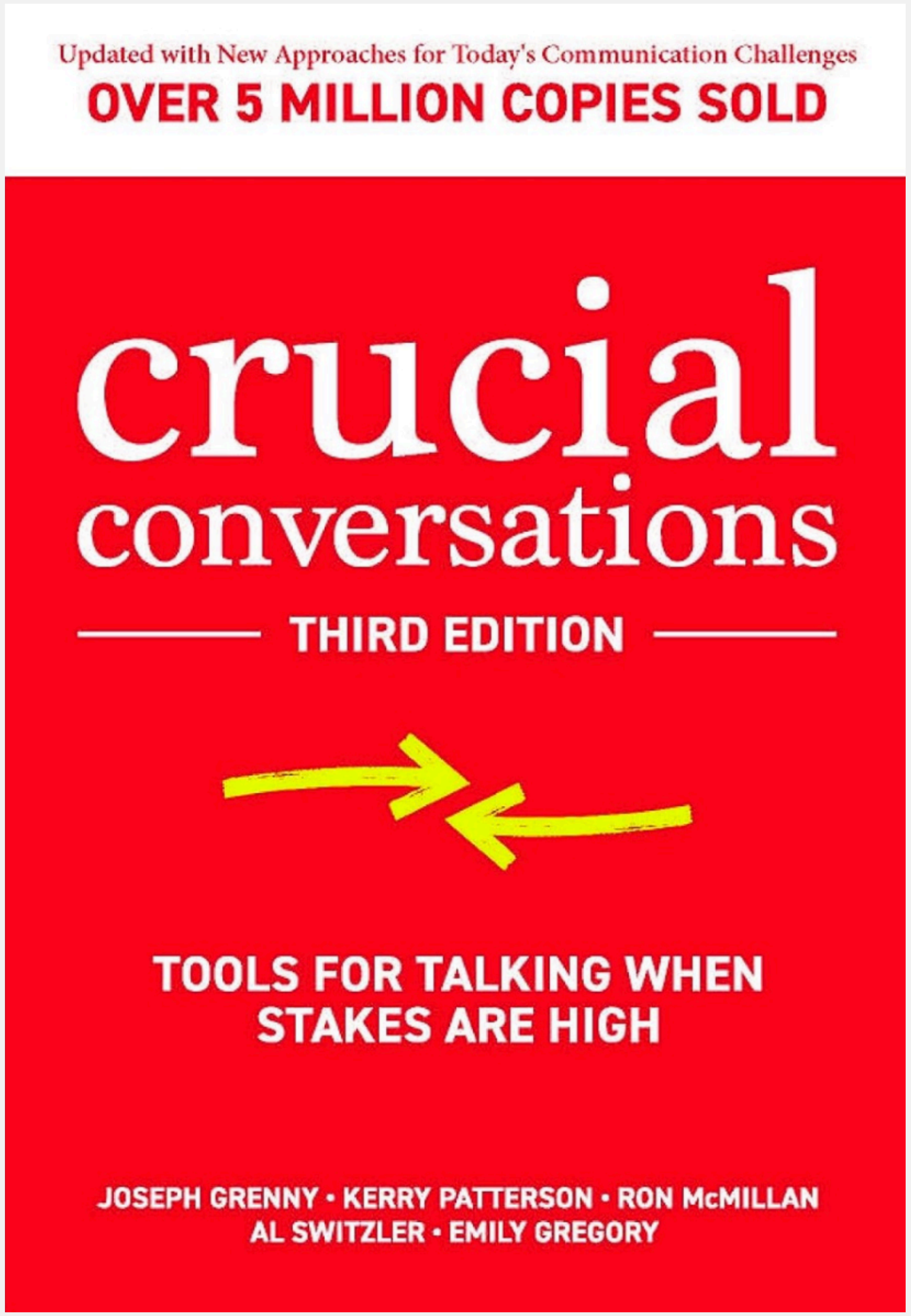
748 Accesses | **19** Citations | **1** Altmetric | Metrics

Abstract

Many recent software engineering papers have examined duplicate issue reports. Thus far, duplicate reports have been considered a hindrance to developers and a drain on their resources. As a result, prior research in this area focuses on proposing automated approaches to accurately identify duplicate reports. However, there exists no studies that attempt to

Resolving Conflicts

Resolving Conflicts



Updated with New Approaches for Today's Communication Challenges

OVER 5 MILLION COPIES SOLD

crucial conversations

THIRD EDITION

TOOLS FOR TALKING WHEN STAKES ARE HIGH

JOSEPH GRENNY • KERRY PATTERSON • RON McMILLAN
AL SWITZLER • EMILY GREGORY

Communication!

Communication

Communication

You can't solve any Problem without Communication!

Communication

Communication

Conflict Resolution

- Your goal: Find a solution to the problem and move forward.
  - As a smart person on ”TedLasso” once said,“Fight forward,not back.”
- Make sure that everybody works from the same set of facts.
- Establish ground rules for your team’s discussion.
  - Talk about how the situation made you feel.Never presume anything about anyone else.
- Remain calm and rational. If you feel triggered or threatened, extract yourself from the situation, wait an hour to chill out, and then try again.
- If you reach an impasse, talk to your team leader.
- If your team remains in conflict, escalate to Dr. Moran.
  - I can help to mediate

Software Testing

Learning Goals

- Identify the scope and limitations of software testing
- Appreciate software testing as a methodology to use automation in improving software quality
- Describe the benefits of using continuous integration and deployment (CI/CD)
- Measure the quality of software tests and define test adequacy criteria
- Enumerate different levels of testing such as unit testing, integration testing, system testing, and testing in production
- Describe the principles of test-driven development
- Outline design principles for writing good tests
- Recognize and avoid testing anti-patterns

What is Testing Good For?

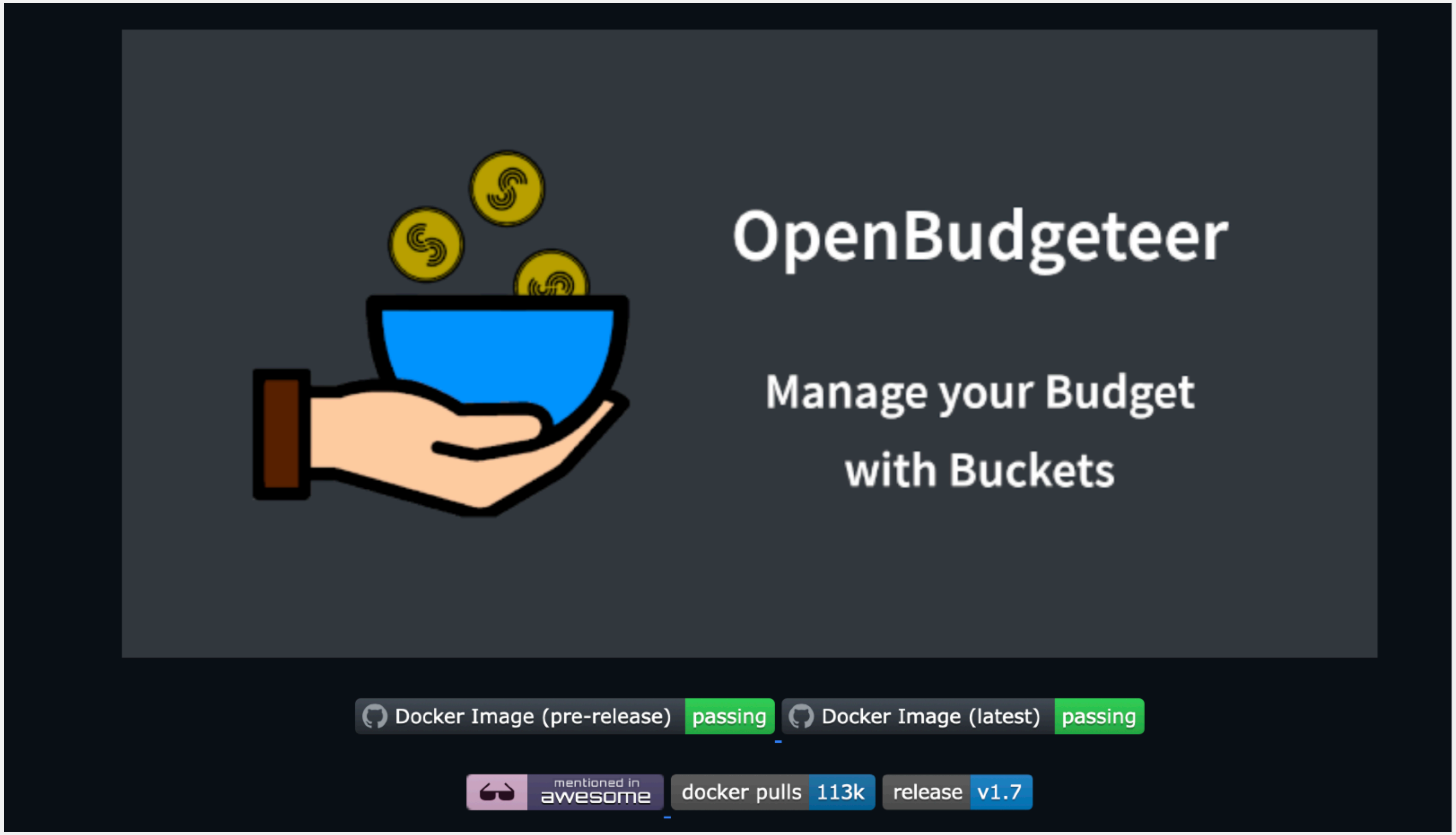
- What is testing?
  - Execution of code on sample inputs in a controlled environment
- Principle goals:
  - Validation: program meets requirements, including quality attributes.
  - Defect testing: reveal failures.

What is Testing Good For?

- Why should we test? What does testing achieve?
  - What does testing not achieve?
- When should we test?
  - And where should we run the tests?
- What should we test?
  - What CAN we test? (Software quality attributes)
- How should we test?
  - How many ways can you test the sort() function?
- How good are our tests?
  - How to measure test quality?

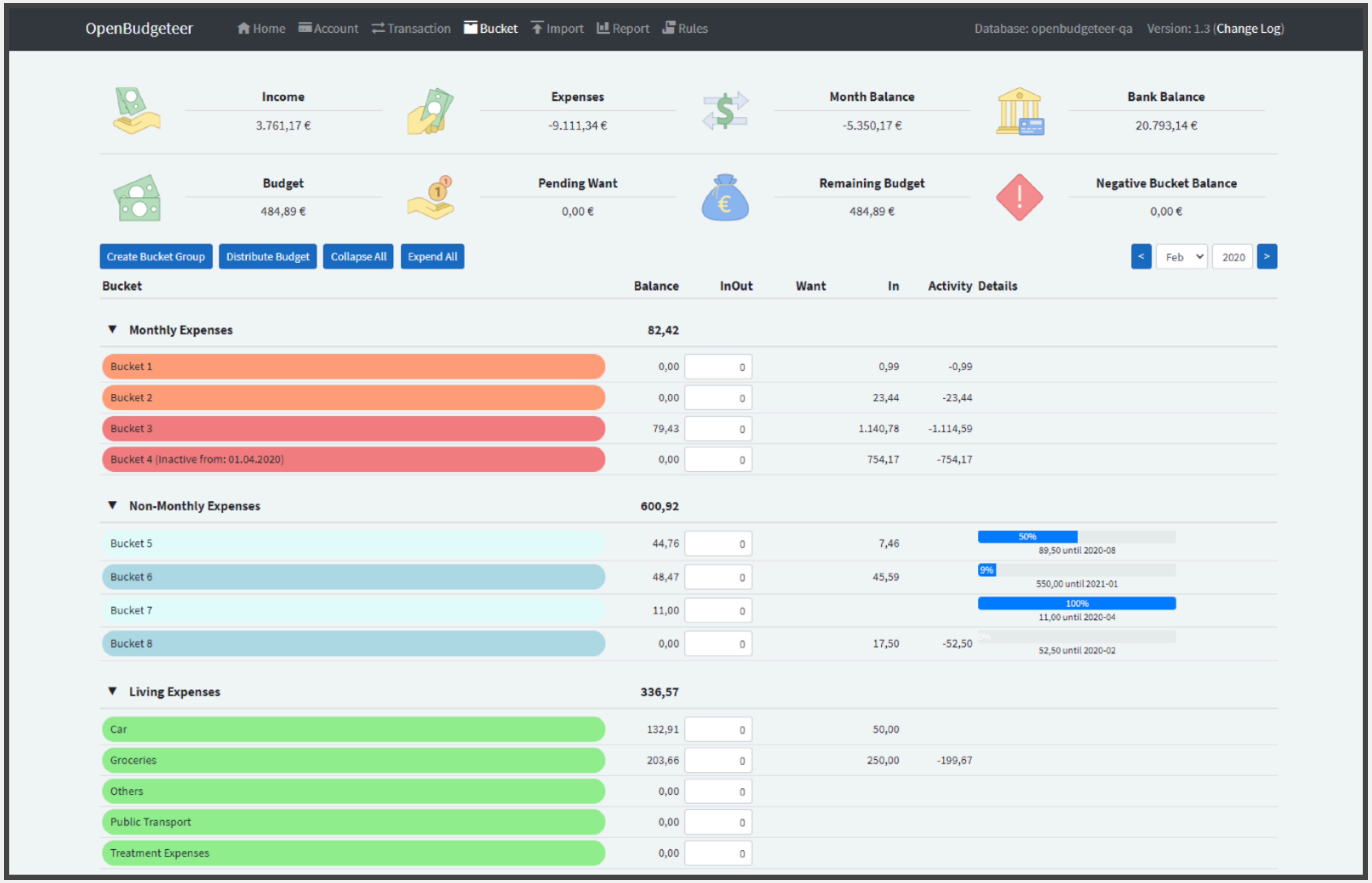
What Makes a Good Test?

What Makes a Good Test?



https://github.com/TheAxelander/OpenBudgeteer

What Makes a Good Test?



Income	3.761,17 €
Expenses	-9.111,34 €
Month Balance	-5.350,17 €
Bank Balance	20.793,14 €
Budget	484,89 €
Pending Want	0,00 €
Remaining Budget	484,89 €
Negative Bucket Balance	0,00 €

Bucket	Balance	InOut	Want	In	Activity	Details
▼ Monthly Expenses	82,42					
Bucket 1	0,00	0		0,99	-0,99	
Bucket 2	0,00	0		23,44	-23,44	
Bucket 3	79,43	0		1.140,78	-1.114,59	
Bucket 4 (Inactive from: 01.04.2020)	0,00	0		754,17	-754,17	
▼ Non-Monthly Expenses	600,92					
Bucket 5	44,76	0		7,46		50% 89,50 until 2020-08
Bucket 6	48,47	0		45,59		9% 550,00 until 2021-01
Bucket 7	11,00	0				100% 11,00 until 2020-04
Bucket 8	0,00	0		17,50	-52,50	0% 52,50 until 2020-02
▼ Living Expenses	336,57					
Car	132,91	0		50,00		
Groceries	203,66	0		250,00	-199,67	
Others	0,00	0				
Public Transport	0,00	0				
Treatment Expenses	0,00	0				

https://github.com/TheAxelander/OpenBudgeteer

Why Write Tests at All?

- [Low bar] Ensure that our software meets requirements, is correct, etc.
- Preventing bugs or quality degradations from being accidentally introduced in the future -> *Regression Testing*
- Helps uncover unexpected behaviors that can't be identified by reading source code
- Increased confidence in changes ("will I break the internet with this commit?")
- Bridges the gap between a declarative view of the system (i.e., requirements) and an imperative view (i.e., implementation) by means of redundancy.
- Tests are executable documentation; increases code maintainability
- Forces writing testable code <-> checks software design

Testing Levels

- Unit testing
  - Code level, E.g. is a function implemented correctly?
  - Does not require setting up a complex environment
- Integration testing
  - Do components interact correctly? E.g. a feature that cuts across client and server.
  - Usually requires some environment setup, but can abstract/mock out other components that are not being tested (e.g. network)
- System testing
  - Validating the whole system end-to-end (E2E)
  - Requires complete deployment in a staging area, but fake data
- Testing in production
  - Real data but more risks

What are the Limitations of Testing?

- "*Testing shows the presence, not the absence of bugs.*" - Edsger W. Dijkstra
- Testing doesn't really give any formal assurances
- Writing tests is hard, time consuming
- Knowing if your tests are good enough is not obvious
- Executing tests can be expensive, especially as software complexity and configuration space grows
    - Full test suite for a single large app can take several days to run

What can We Test for?

Test Oracles

- “Oracles” are mechanisms that tell you when program execution seems abnormal or unexpected
- E.g. assert, segfault, exception
- Other examples: performance threshold, memory footprint, address sanitizer

Test Oracles

- Obvious in some applications (e.g. “sort()”) but more challenging in others (e.g. “encrypt()” or UI-based tests)
- Lack of good oracles can limit the scalability of testing. Easy to generate lots of input data, but not easy to validate if output (or other program behavior) is correct.
- Fortunately, we have some tricks.

Differential Testing

- If you have two implementations of the same specification, then their output should match on all inputs.
  - E.g. `mergeSort(x).equals(bubbleSort(x))` -> should always be true
  - Special case of a property test, with a free oracle.

- If a differential test fails, at least one of the two implementations is wrong.
  - But which one?
  - If you have N>2 implementations, run them all and compare. Majority wins (the odd one out is buggy).

- Differential testing works well when testing programs that implement standard specifications such as compilers, browsers, SQL engines, XML/JSON parsers, media players, etc.
  - Not feasible in general e.g. for UCF’s custom grad application system.

Regression Testing

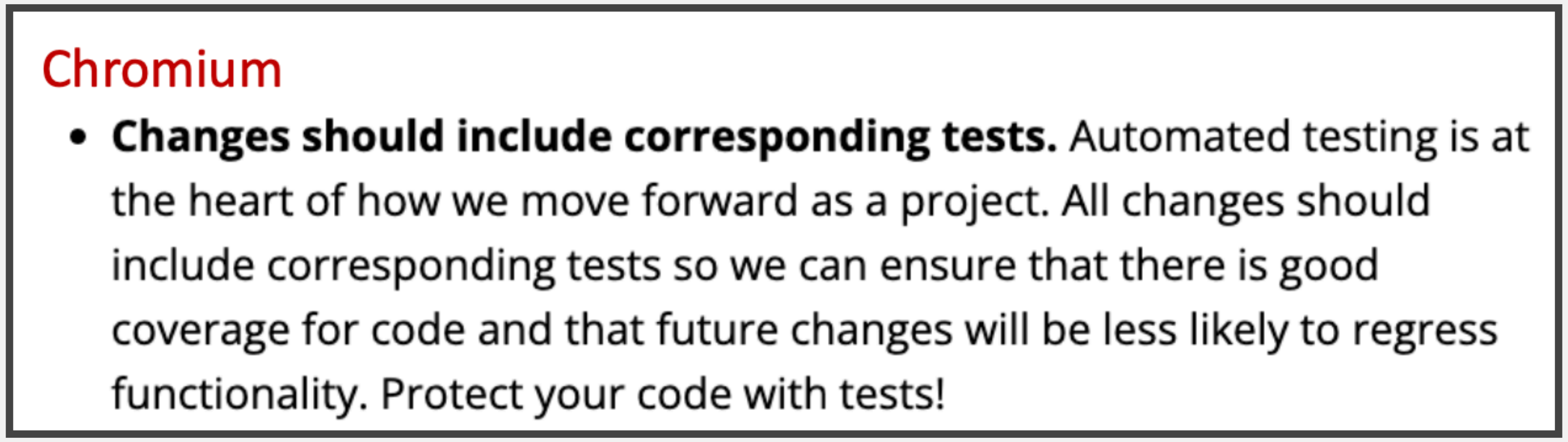
- Differential testing through time (or versions, say V1 and V2).
- Assuming V1 and V2 don't add a new feature or fix a known bug, then f(x) in V1 should give the same result as f(x) in V2.
- *Key Idea:* Assume the current version is correct. Run program on current version and log output. Compare all future versions to that output.

When Should We Test?

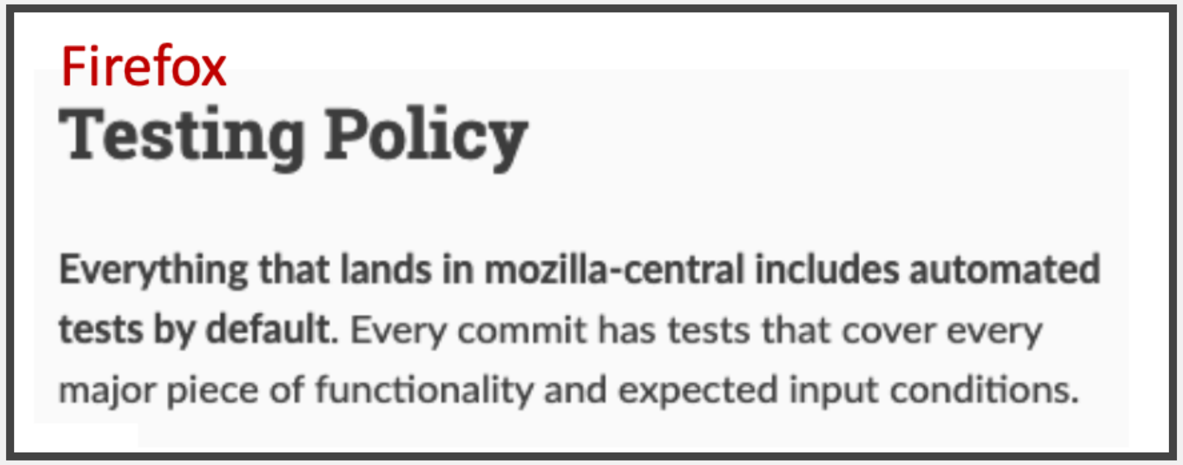
Test Driven Development

- Tests first!
- Popular agile technique
- Write tests as specifications before code
- Never write code without a failing test
- Claims:
  - Design approach toward testable design
  - Think about interfaces first
  - Avoid unneeded code
  - Higher product quality
  - Higher test suite quality
  - Higher overall productivity

Common Bar for Contributions

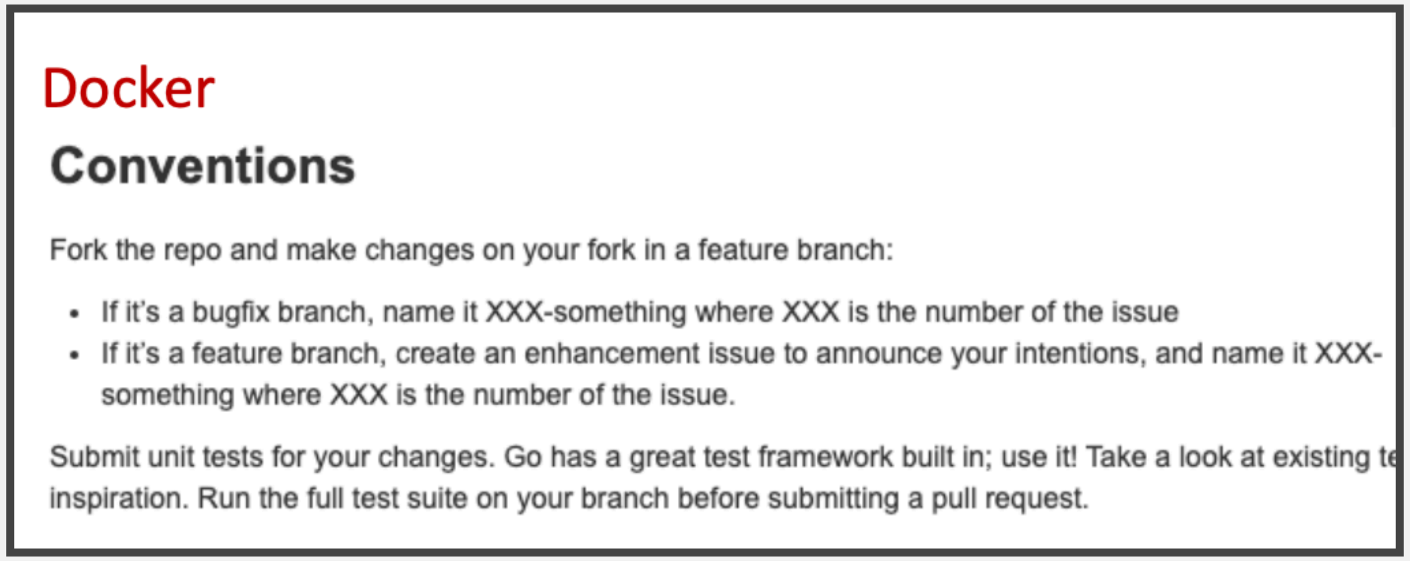

Chromium

- **Changes should include corresponding tests.** Automated testing is at the heart of how we move forward as a project. All changes should include corresponding tests so we can ensure that there is good coverage for code and that future changes will be less likely to regress functionality. Protect your code with tests!


Firefox

Testing Policy

Everything that lands in mozilla-central includes automated tests by default. Every commit has tests that cover every major piece of functionality and expected input conditions.


Docker

Conventions

Fork the repo and make changes on your fork in a feature branch:

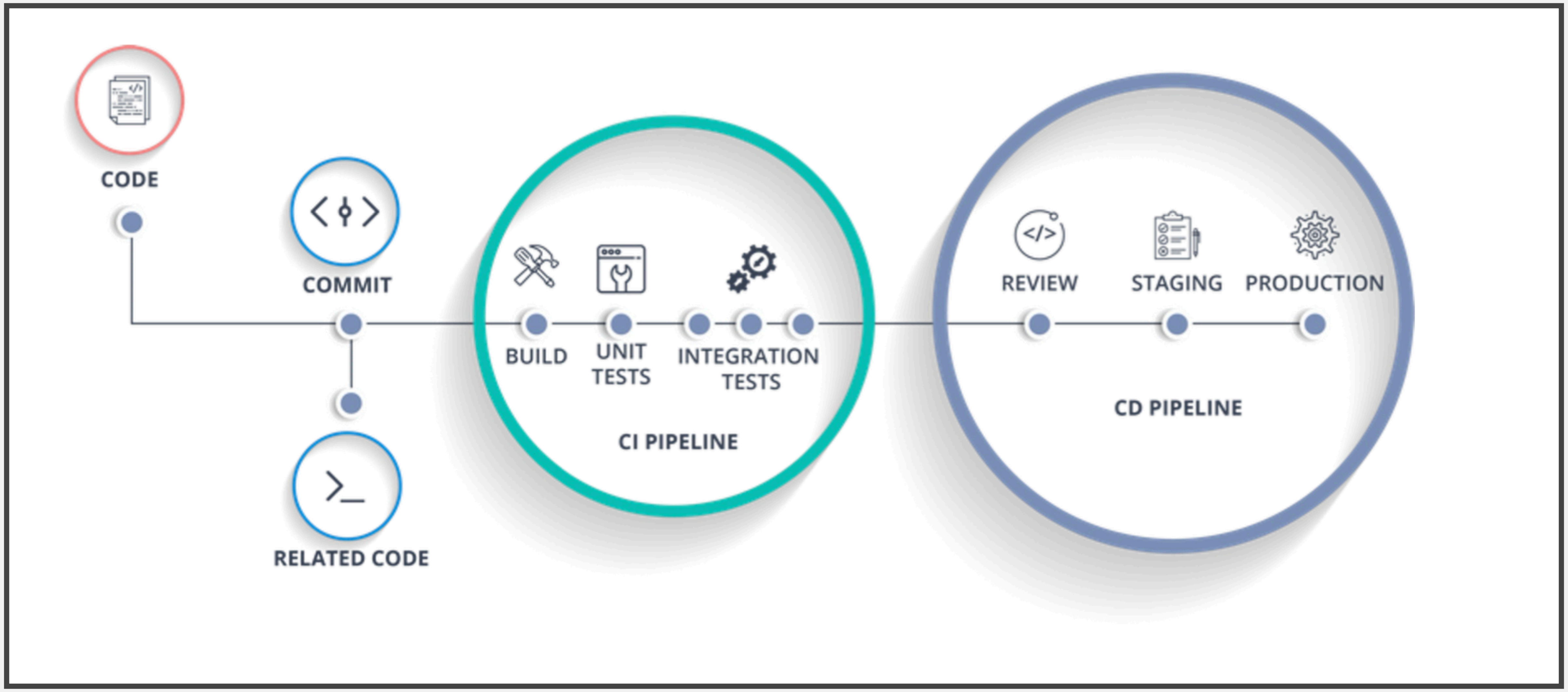
- If it's a bugfix branch, name it XXX-something where XXX is the number of the issue
- If it's a feature branch, create an enhancement issue to announce your intentions, and name it XXX-something where XXX is the number of the issue.

Submit unit tests for your changes. Go has a great test framework built in; use it! Take a look at existing te
inspiration. Run the full test suite on your branch before submitting a pull request.

Regression Testing

- Usual model:
    - Introduce regression tests for bug fixes, etc.
    - Compare results as code evolves
        - Code1 + TestSet -> TestResults1
        - Code2 + TestSet -> TestResults2
    - As code evolves, compare TestResults1 with TestResults2, etc.
- Benefits:
- Ensure bug fixes remain in place and bugs do not reappear.
- Reduces reliance on specifications, as <TestSet,TestResults1> acts as one.

Continuous Integration & Deployment

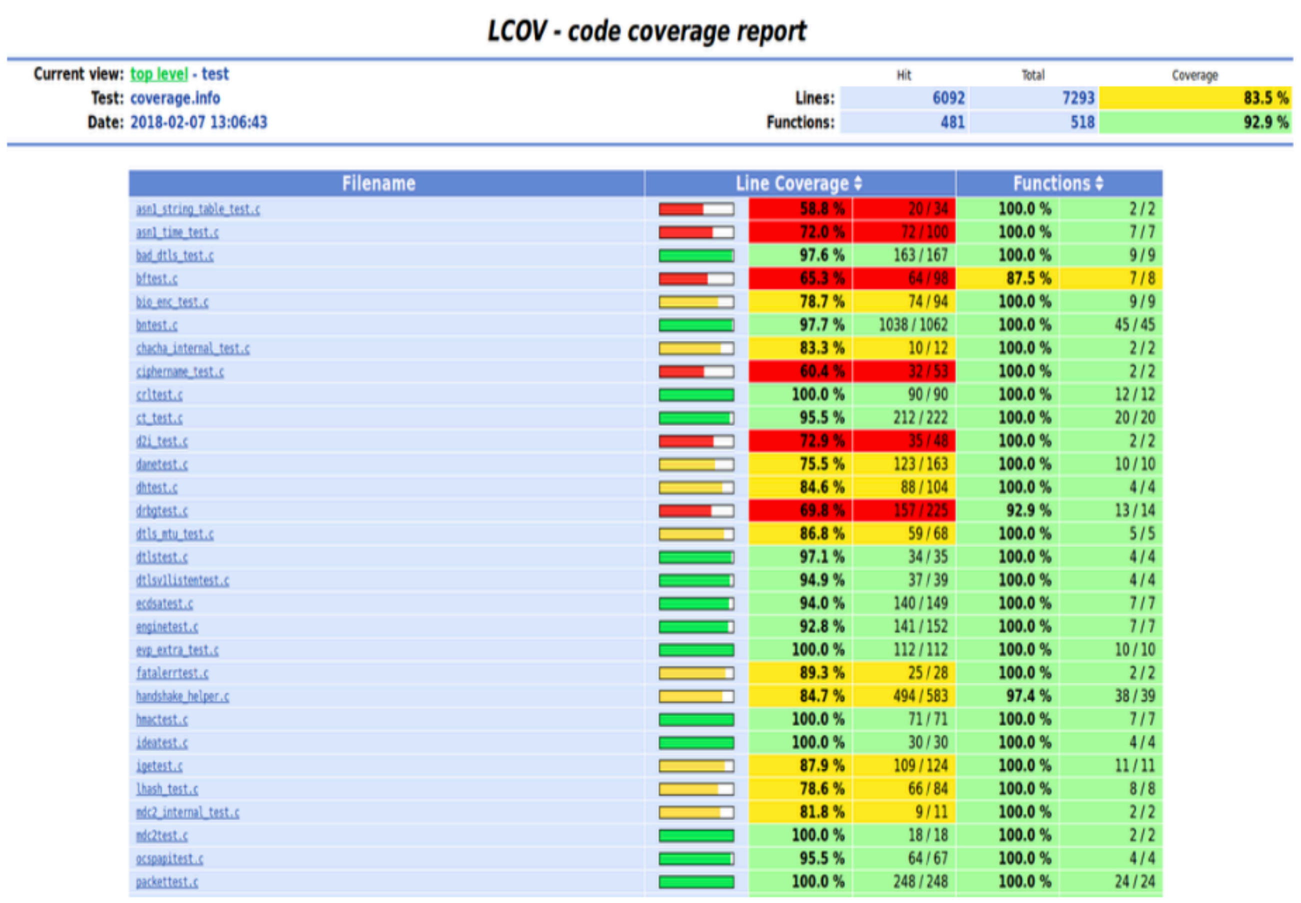


How Good Are Our Tests?

Code Coverage

- Line coverage
  - Statement coverage
  - Branch coverage
  - Instruction coverage
  - Basic-block coverage
  - Edge coverage
  - Path coverage
  - ...

Code Coverage

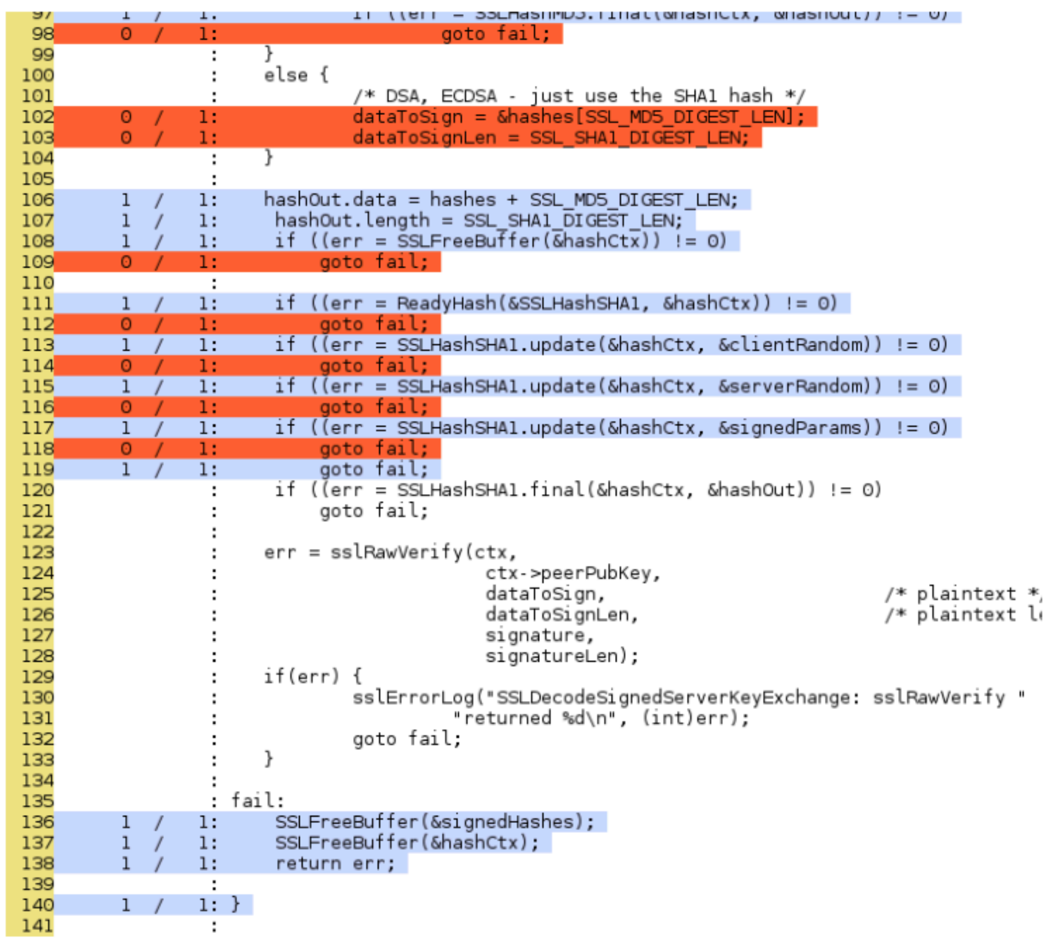


LCOV - code coverage report

Current view: top level - test
Test: coverage.info
Date: 2018-02-07 13:06:43

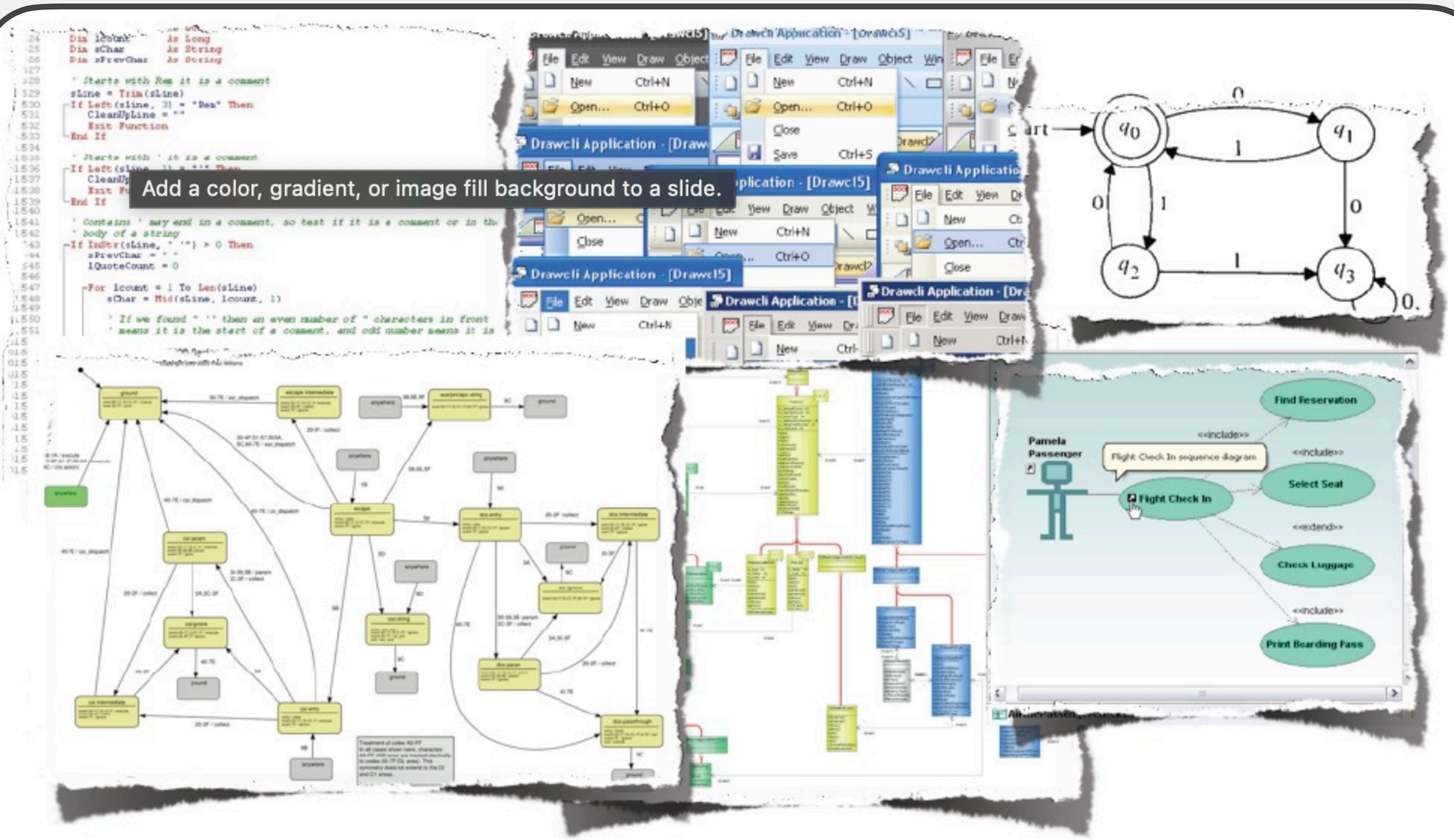
	Hit	Total	Coverage
Lines:	6092	7293	83.5 %
Functions:	481	518	92.9 %

Filename	Line Coverage ⇕			Functions ⇕	
asn1_string_table_test.c	🟥	58.8 %	20 / 34	100.0 %	2 / 2
asn1_time_test.c	🟥	72.0 %	72 / 100	100.0 %	7 / 7
bad_dtls_test.c	🟩	97.6 %	163 / 167	100.0 %	9 / 9
bftest.c	🟥	65.3 %	64 / 98	87.5 %	7 / 8
bio_enc_test.c	🟨	78.7 %	74 / 94	100.0 %	9 / 9
bntest.c	🟩	97.7 %	1038 / 1062	100.0 %	45 / 45
chacha_internal_test.c	🟨	83.3 %	10 / 12	100.0 %	2 / 2
ciphername_test.c	🟥	60.4 %	32 / 53	100.0 %	2 / 2
crltest.c	🟩	100.0 %	90 / 90	100.0 %	12 / 12
ct_test.c	🟩	95.5 %	212 / 222	100.0 %	20 / 20
d2i_test.c	🟥	72.9 %	35 / 48	100.0 %	2 / 2
danetest.c	🟨	75.5 %	123 / 163	100.0 %	10 / 10
dhtest.c	🟨	84.6 %	88 / 104	100.0 %	4 / 4
drbgtest.c	🟥	69.8 %	157 / 225	92.9 %	13 / 14
dtls_mtu_test.c	🟨	86.8 %	59 / 68	100.0 %	5 / 5
dtlstest.c	🟩	97.1 %	34 / 35	100.0 %	4 / 4
dtlsv1listentest.c	🟩	94.9 %	37 / 39	100.0 %	4 / 4
ecdsatest.c	🟩	94.0 %	140 / 149	100.0 %	7 / 7
enginetest.c	🟩	92.8 %	141 / 152	100.0 %	7 / 7
evp_extra_test.c	🟩	100.0 %	112 / 112	100.0 %	10 / 10
fatalerrtest.c	🟨	89.3 %	25 / 28	100.0 %	2 / 2
handshake_helper.c	🟨	84.7 %	494 / 583	97.4 %	38 / 39
hmactest.c	🟩	100.0 %	71 / 71	100.0 %	7 / 7
ideatest.c	🟩	100.0 %	30 / 30	100.0 %	4 / 4
igetest.c	🟨	87.9 %	109 / 124	100.0 %	11 / 11
lhash_test.c	🟨	78.6 %	66 / 84	100.0 %	8 / 8
mdc2_internal_test.c	🟨	81.8 %	9 / 11	100.0 %	2 / 2
mdc2test.c	🟩	100.0 %	18 / 18	100.0 %	2 / 2
ocspapitest.c	🟩	95.5 %	64 / 67	100.0 %	4 / 4
packettest.c	🟩	100.0 %	248 / 248	100.0 %	24 / 24



```
     1 /   1:             if ((err = SSLHashMD5.final(&hashCtx, &hashOut)) != 0)
     0 /   1:                 goto fail;
           :     }
           :     else {
           :             /* DSA, ECDSA - just use the SHA1 hash */
     0 /   1:             dataToSign = &hashes[SSL_MD5_DIGEST_LEN];
     0 /   1:             dataToSignLen = SSL_SHA1_DIGEST_LEN;
           :     }
           :
     1 /   1:     hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
     1 /   1:      hashOut.length = SSL_SHA1_DIGEST_LEN;
     1 /   1:      if ((err = SSLFreeBuffer(&hashCtx)) != 0)
     0 /   1:          goto fail;
           :
     1 /   1:      if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
     0 /   1:          goto fail;
     1 /   1:      if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
     0 /   1:          goto fail;
     1 /   1:      if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
     0 /   1:          goto fail;
     1 /   1:      if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
     0 /   1:          goto fail;
     1 /   1:          goto fail;
           :      if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
           :          goto fail;
           :
           :     err = sslRawVerify(ctx,
           :                        ctx->peerPubKey,
           :                        dataToSign,                             /* plaintext *
           :                        dataToSignLen,                          /* plaintext l
           :                        signature,
           :                        signatureLen);
           :     if(err) {
           :             sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
           :                         "returned %d\n", (int)err);
           :             goto fail;
           :     }
           :
           : fail:
     1 /   1:      SSLFreeBuffer(&signedHashes);
     1 /   1:      SSLFreeBuffer(&hashCtx);
     1 /   1:      return err;
           :
     1 /   1: }
           :
```

We Can Measure Coverage on Almost Anything



```
Dim lcount      As Long
Dim sChar       As String
Dim sPrevChar   As String

' Starts with Rem it is a comment
sLine = Trim(sLine)
If Left(sline, 3) = "Rem" Then
   CleanUpLine = ""
   Exit Function
End If

' Starts with ' it is a comment
If Left(sLine, 1) = "'" Then
   CleanUp
   Exit Fu
End If

' Contains ' may end in a comment, so test if it is a comment or in th
' body of a string
If InStr(sLine, " '") > 0 Then
   sPrevChar = " "
   lQuoteCount = 0

   For lcount = 1 To Len(sLine)
      sChar = Mid(sLine, lcount, 1)

      ' If we found " '' then an even number of " characters in front
      ' means it is the start of a comment, and odd number means it is
```

Add a color, gradient, or image fill background to a slide.

Be Aware of Coverage Chasing

- Recall: issues with metrics and incentives
  - Also: Numbers can be deceptive

- 100% coverage != exhaustively tested
  - “Coverage is not strongly correlated with suite effectiveness”

- Based on empirical study on GitHub projects [Inozemtseva and Holmes, ICSE’14]

- Still, it’s a good low bar
  - Code that is not executed has definitely not been tested

Coverage of What?

- Distinguish code being tested and code being executed
- Library code >>>> Application code
  - Can selectively measure coverage
- All application code >>> code being tested
  - Not always easy to do this within an application

Coverage != Outcome

- What's better, tests that always pass or tests that always fail?
- Tests should ideally be falsifiable. Boundary determines
- specification
- Ideally:
  - Correct implementations should pass all tests
  - Buggy code should fail at least one test
  - Intuition behind mutation testing (we'll revisit this next week)
- What if tests have bugs?
  - Pass on buggy code or fail on correct code
- Even worse: flaky tests
  - Pass or fail on the same test case nondeterministically
- What's the worst type of test?

Test Design Principles

- Use public APIs only
- Clearly distinguish inputs, configuration, execution, and oracle
- Be simple; avoid complex control flow such as conditionals and loops
- Tests shouldn't need to be frequently changed or refactored
  - Definitely not as frequently as the code being tested changes

Anti-Patterns

- Snoopy oracles
  - Relying on implementation state instead of observable behavior
  - E.g. Checking variables or fields instead of return values
- Brittle tests
  - Overfitting to special-case behavior instead of general principle
  - E.g. hard-coding message strings instead of behavior
- Slow tests
  - Self-explanatory(beware of heavy environments, I/O, and sleep())
- Flaky tests
  - Tests that pass or fail nondeterministically
  - Often because of reliance on random inputs, timing (e.g. sleep(1000)), availability of external services (e.g. fetching data over the network in a unit test), or dependency on order of test execution (e.g. previous test sets up global variables in certain way)

Takeaways

- Most tests that you will write will be muuuuuuch more complex than testing a sort function.
- Need to set up environment, create objects whose methods to test, create objects for test data, get all these into an interesting state, test multiple APIs with varying arguments, etc.
- Many tests will require mocks (i.e., faking a resource-intensive component).
- General principles of many of these strategies still apply:
  - Writing tests can be time consuming
  - Determining test adequacy can be hard (if not impossible)
  - Test oracles are not easy
  - Advanced test strategies have trade-offs (high costs with high returns)