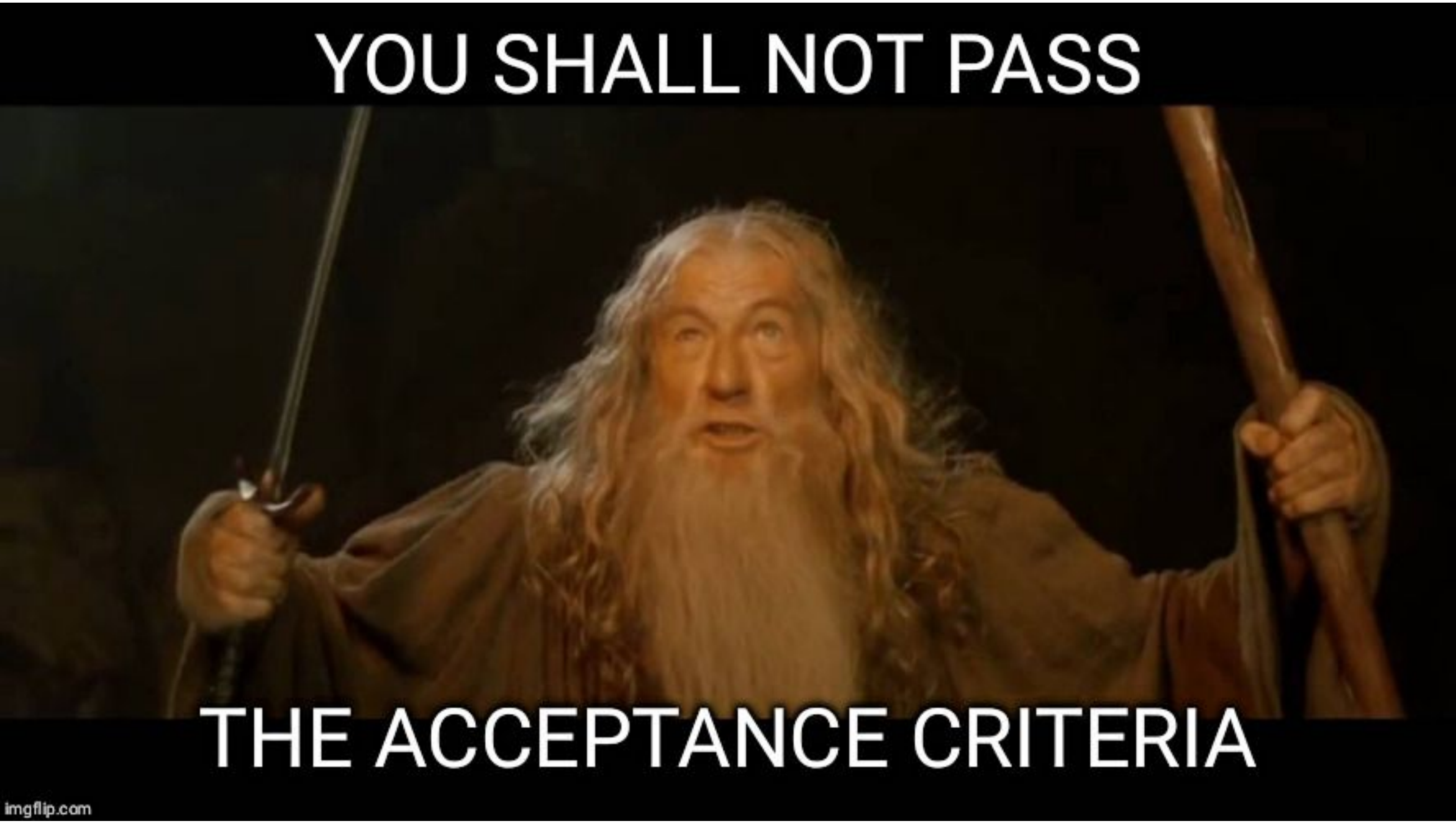
Acceptance Criteria & Technical Debt

Nursultan Askarbekuly


YOU SHALL NOT PASS
THE ACCEPTANCE CRITERIA
imgflip.com

What are Acceptance Criteria?

- Conditions that a software product must be accepted by a user / customer
- A set of statements with a **clear pass/fail result**
- Specifies functional & non-functional requirements
- Applicable at the Epic, Feature, & Story Level

Format

Most popular format is:

Given:	*some precondition*
When:	*I do some action*
Then:	*I expect some result*

User Story 1: Switching app language

As a user **I want to** switch app language **so that** I understand app content.

Given:

When:

Then:

User Story 1: Switching app language

As a user **I want to** switch app language **so that** I understand app content.

Given:
I’m a user on the settings page of the app

When:
I select a language option and confirm switching to that language

Then:
- Current language is changed to the selected language
- App contents correspond to the selected language
- Translation is correct and doesn’t contain critical mistakes
- Translated content does not violate any specific layout

User Story 2: App notifications

As a user **I want to** tune the app’s notifications **so that** I get notified in a way suitable to my needs and schedule.

Given:

When:

Then:

User Story 3: Share with friends and get rewarded

As a user **I want to** tune the app’s notifications **so that** I get notified in a way suitable to my needs and schedule.

Given:
I’m a user on the notifications page of the app

When:
I start configuring the notification options

Then: I should be able to customize the notifications channel, choose suitable sound for push notifications, configure repetition rate and set the appropriate notifications time.

User Story 3: Share with friends and get rewarded

As a user **I want to** be able to recommend the app to my friends **so that** I can get reward points.

Given:

When:

Then:

User Story 3: Share app

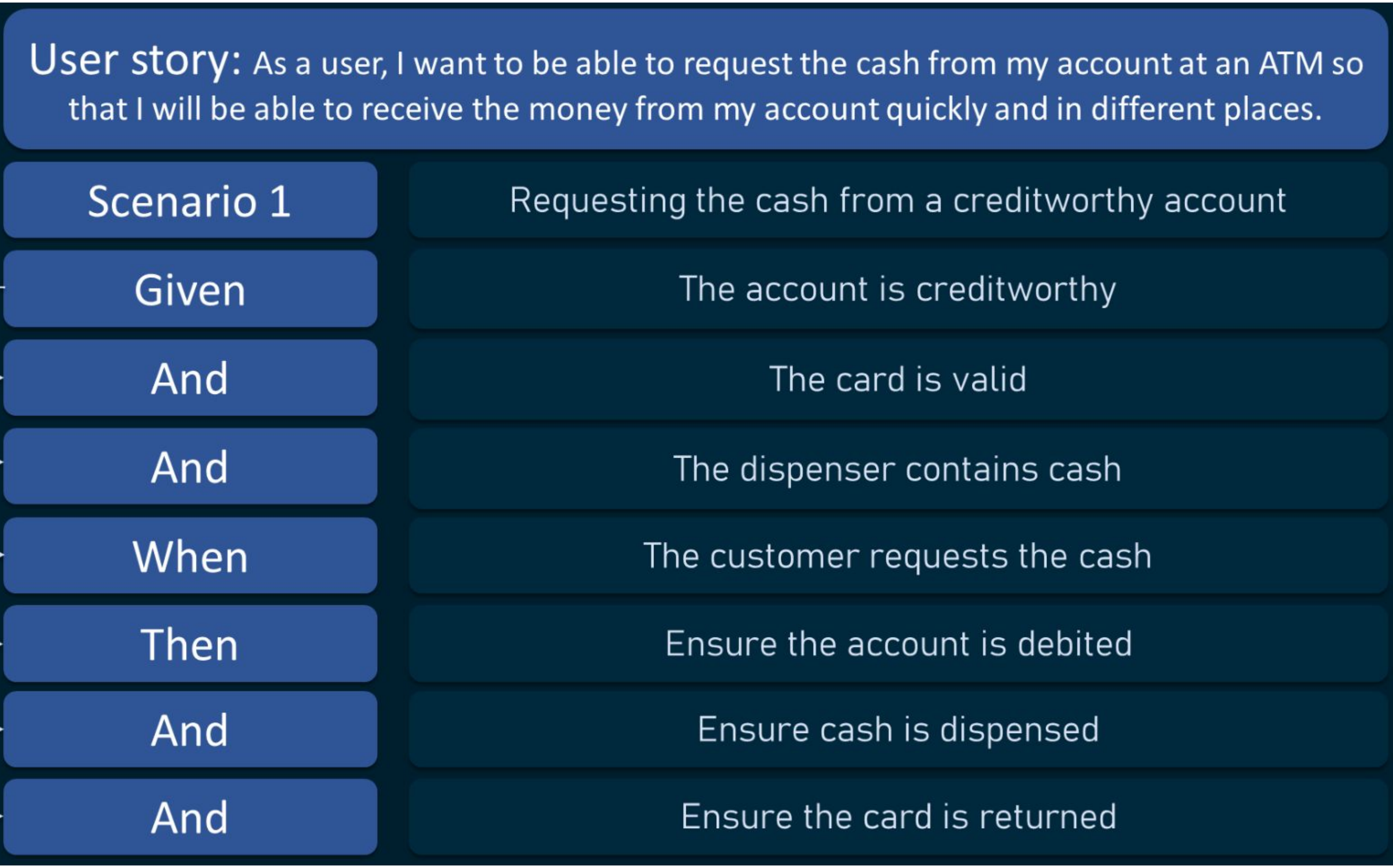
As a user **I want to** be able to recommend the app to my friends **so that** can get reward points.

Given:
I’m a logged in user on the main page of the app

When:
I press share and choose a friend to share the app with

Then:
- The friend receives the installation link
- When the friend installs the app through the sent installation link and registers, I get the correct amount of reward points immediately.

A more complex example



User story: As a user, I want to be able to request the cash from my account at an ATM so that I will be able to receive the money from my account quickly and in different places.

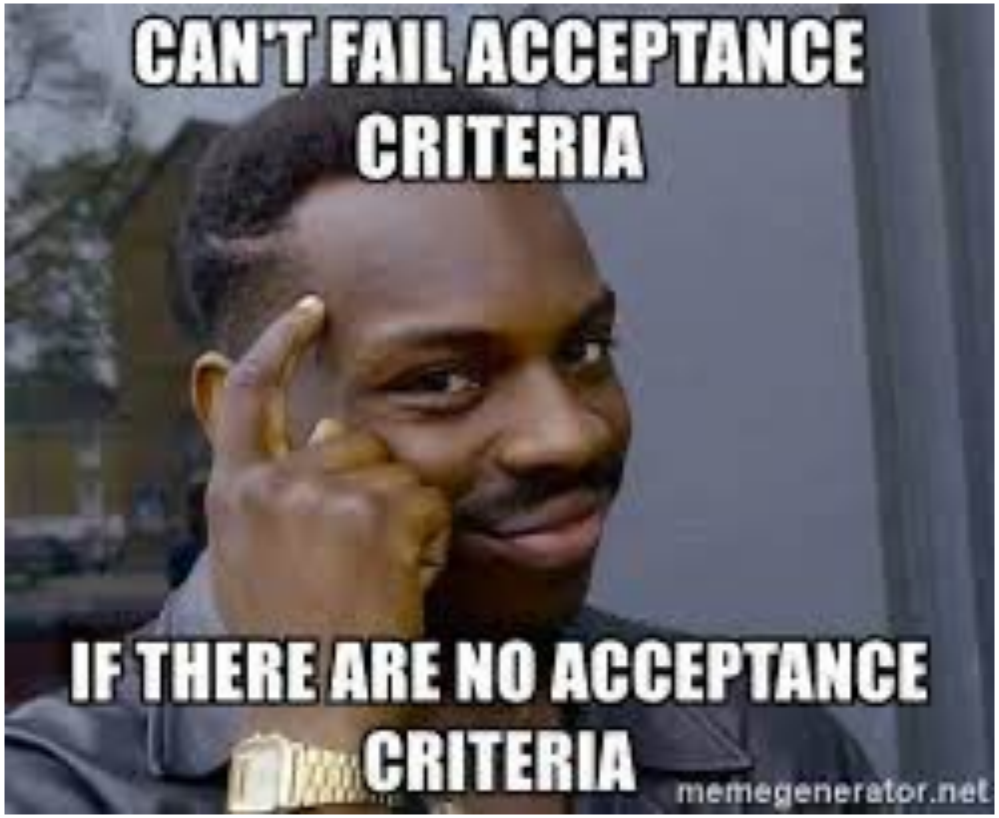
Scenario 1	Requesting the cash from a creditworthy account
Given	The account is creditworthy
And	The card is valid
And	The dispenser contains cash
When	The customer requests the cash
Then	Ensure the account is debited
And	Ensure cash is dispensed
And	Ensure the card is returned

https://www.altexsoft.com/blog/business/acceptance-criteria-purposes-formats-and-best-practices/

- Using the template provides a consistent structure.
- Helps testers determine when to begin and end testing for the work item.

What makes good Acceptance Criteria?

- Defines when a work item is complete & works as expected
- Expresses criteria clearly, in simple language the customer would use
- Allows transformation into automated tests


CAN'T FAIL ACCEPTANCE CRITERIA
IF THERE ARE NO ACCEPTANCE CRITERIA
memegenerator.net

WHAT or HOW?

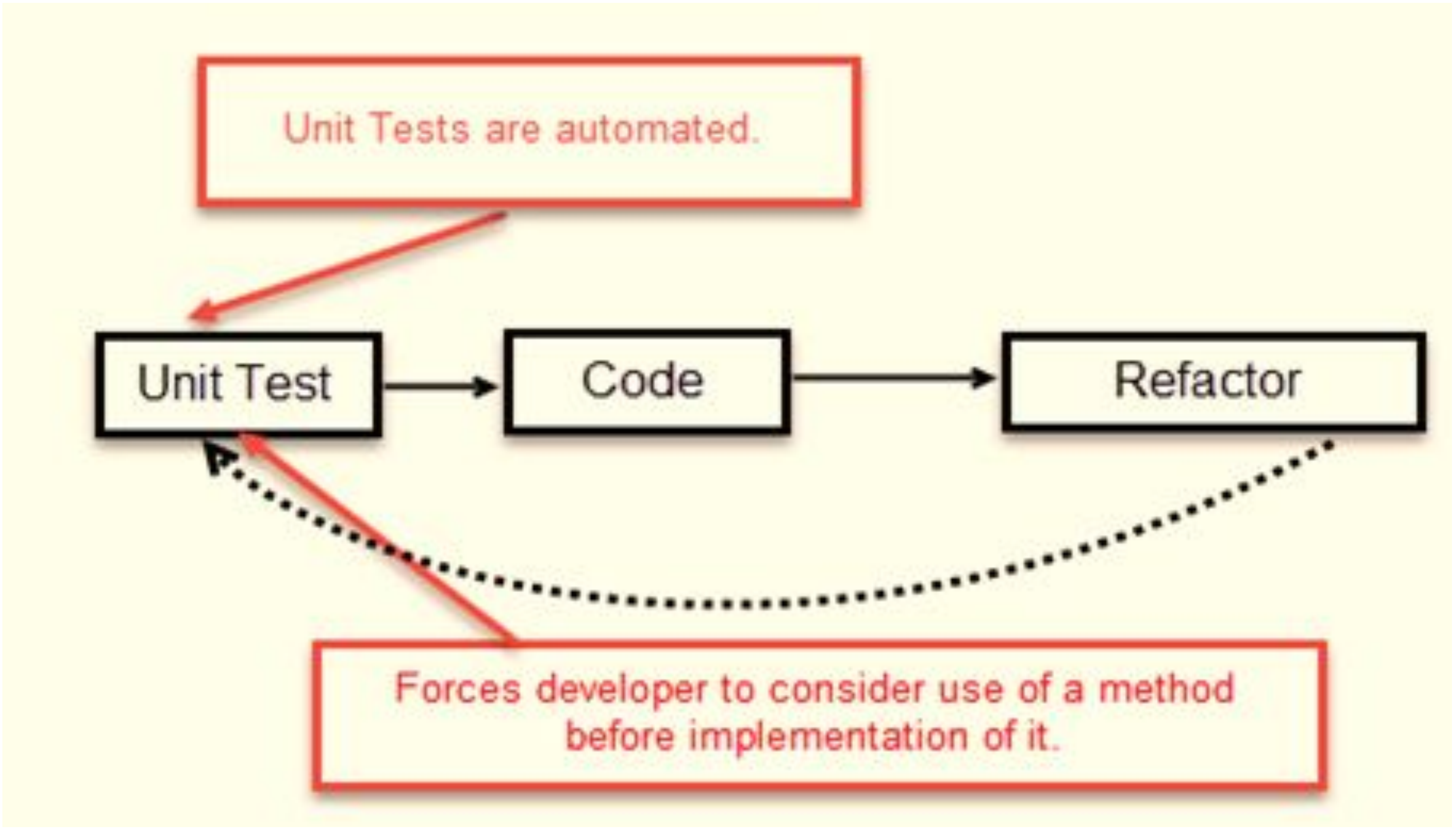
“User can approve or reject an invoice”
“User can click a checkbox to approve an invoice”

WHAT.
Not HOW.

- Criteria should state intent, not a solution.
- The criteria should be independent of the implementation
- Discuss WHAT to expect, and not HOW to implement it

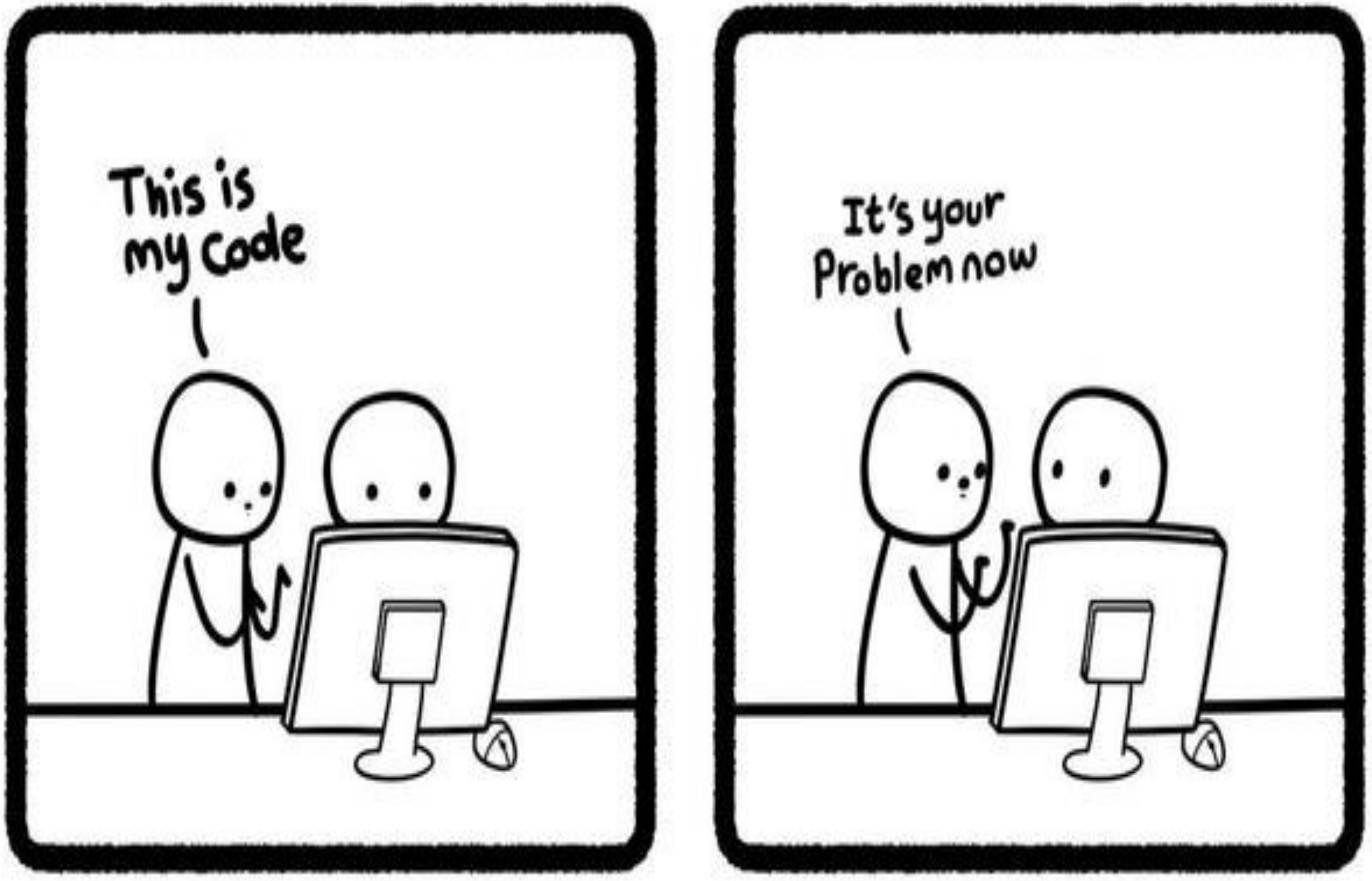
When to define Acceptance Criteria?

Test Driven Development


Unit Tests are automated.
Unit Test
Code
Refactor
Forces developer to consider use of a method before implementation of it.

Technical Debt

Writing bad code: me and future me



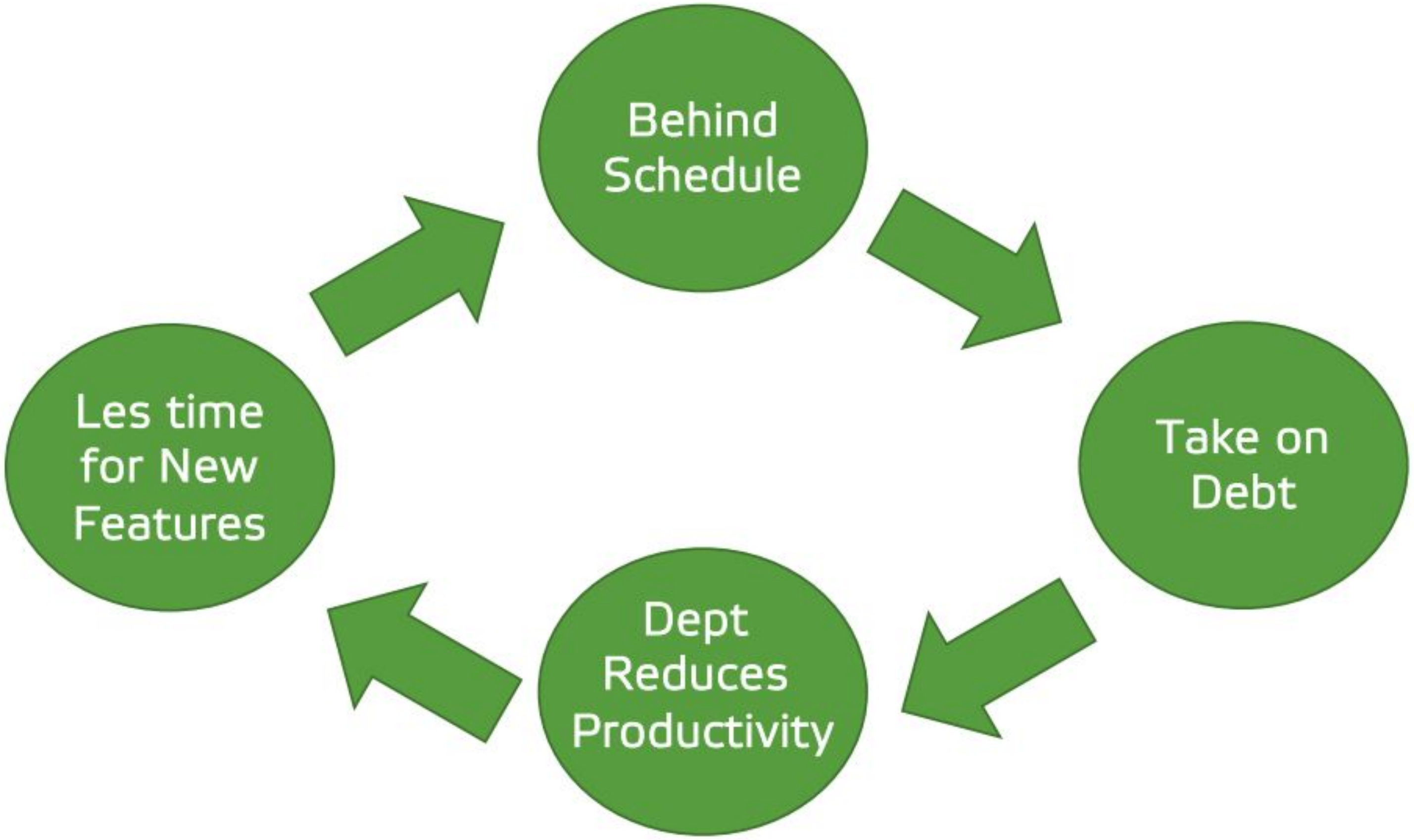
https://twitter.com/nixcraft/status/1268141096908935168

Every Software Engineer Has Been There

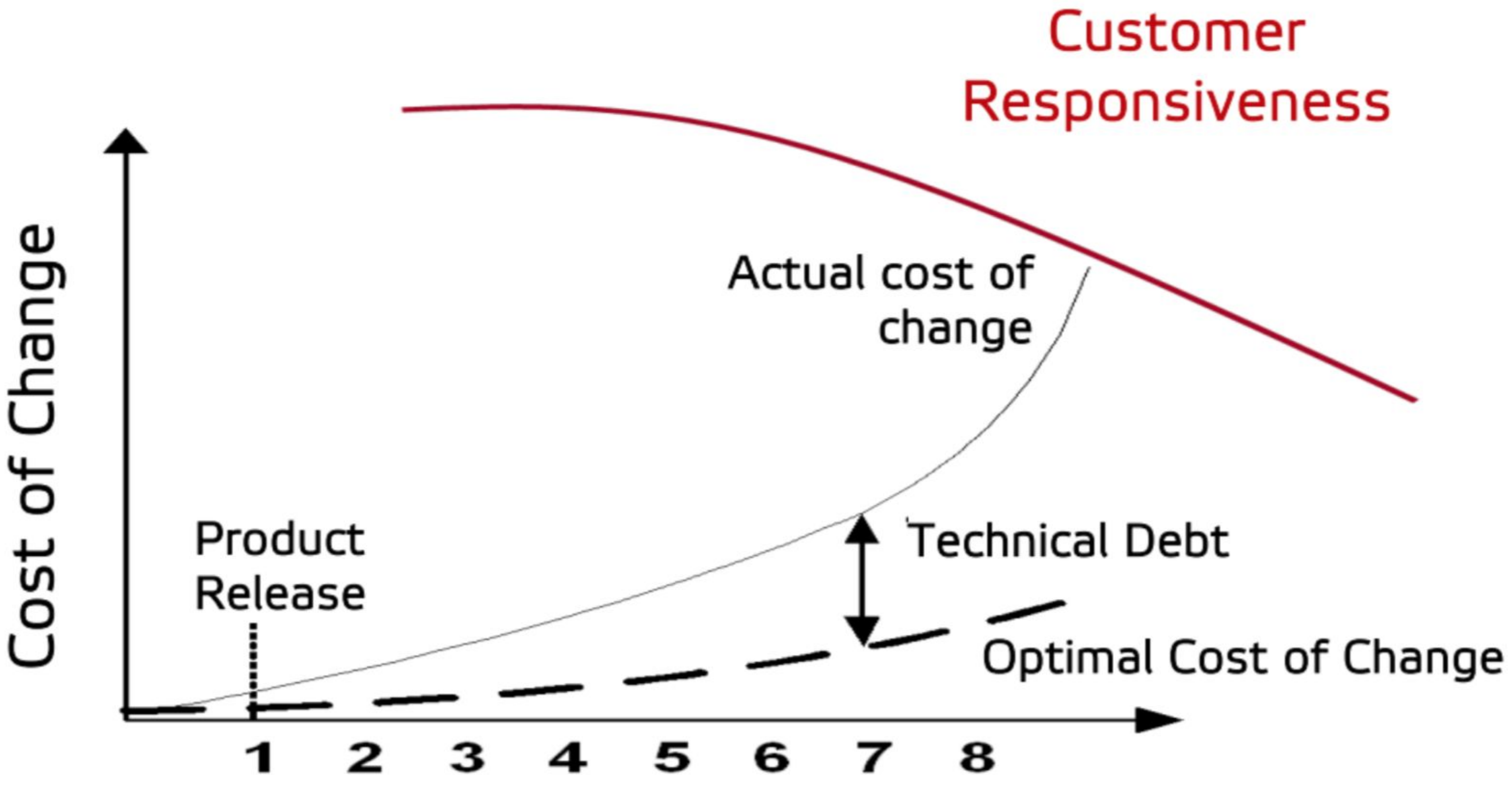
“Guys, we don’t have time to dot every i and cross every t on this release. Just get the code done. It doesn’t have to be perfect. We’ll fix it after we release”

“We don't have time to reconcile these two databases before our deadline, so we'll write some glue code that keeps them synchronized for now and reconcile them after we ship.”

The Technical Debt Cycle



Cost of change over time



Should software do what it is supposed to do or be changeable?

What if you have to choose one?

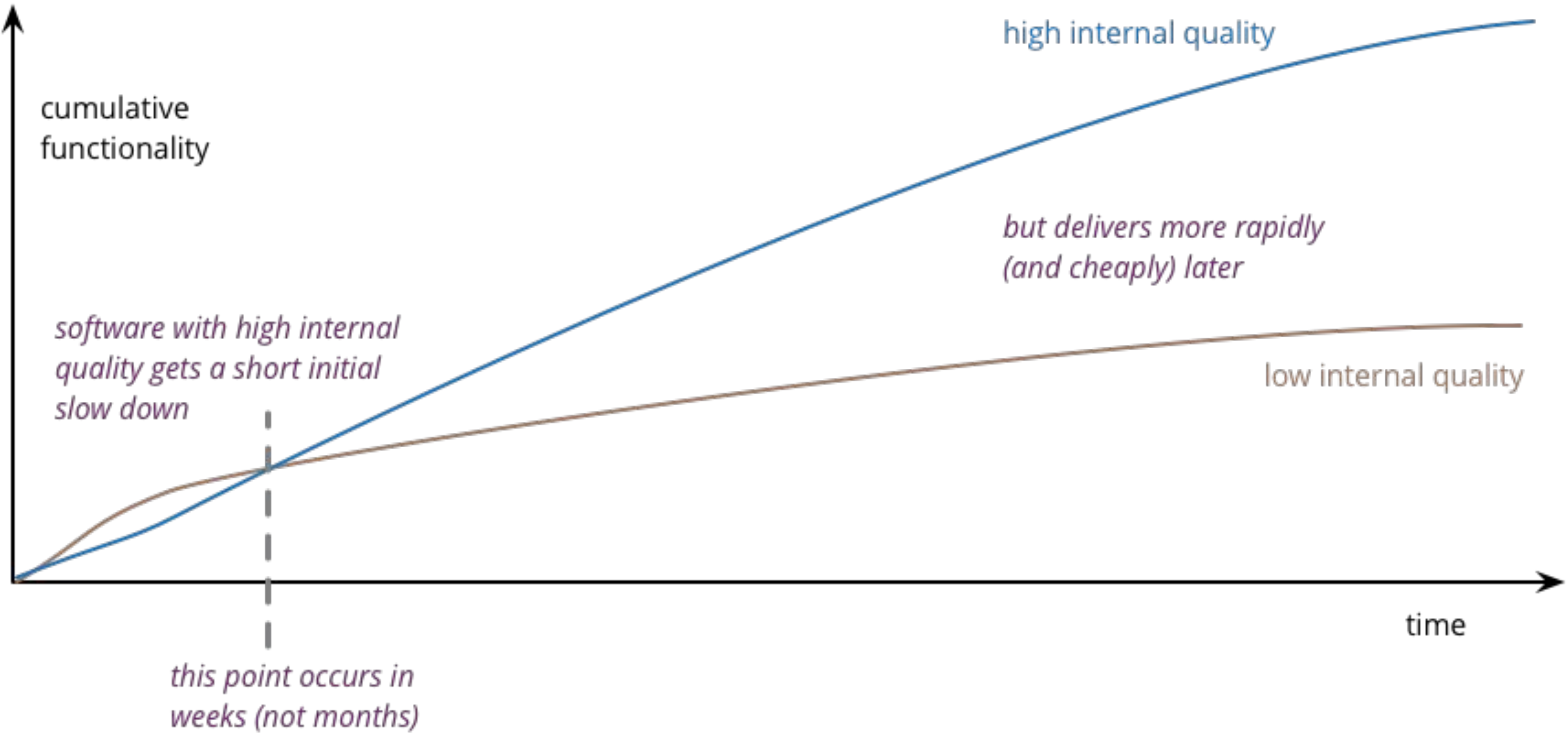
Lehman’s Laws of Program Evolution

Law of continuous change

A large program that is used undergoes continuous change or becomes progressively less useful. The change process continues until it is judged more cost-effective to replace the system with a recreated version.

Law of increasing complexity

A program that is changed, becomes less and less structured (the entropy increases) and thus becomes more complex. One has to invest extra effort in order to avoid increasing complexity.



https://martinfowler.com/articles/is-quality-worth-cost.html

Technical debt quadrant, 2009

	Reckless	Prudent
Deliberate	*“We don’t have time for design”*	*“We must ship now and deal with consequences”*
Inadvertent	*“What’s Layering?”*	*“Now we know how we should have done it”*

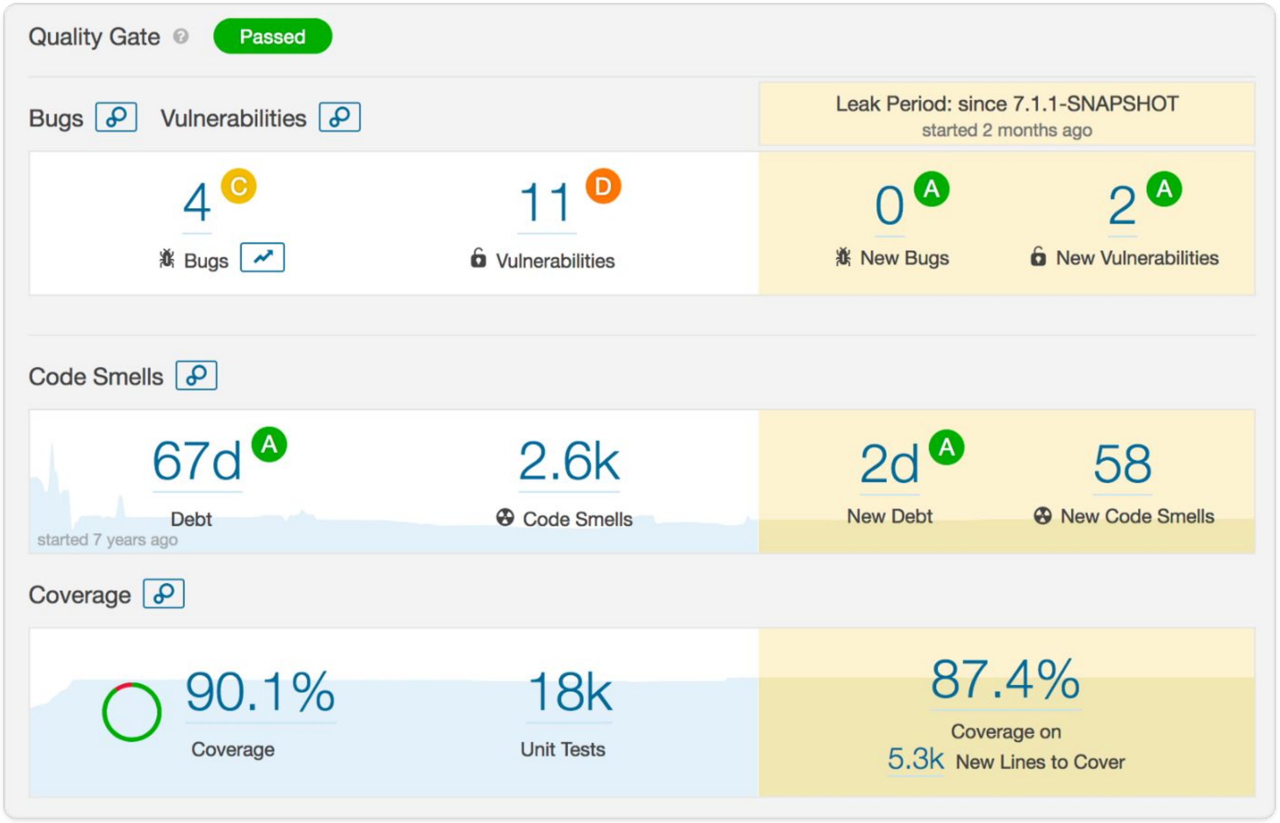
How to manage technical debt?

Tools

Code review

Balance

sonarqube


Quality Gate Passed

Bugs Vulnerabilities

Leak Period: since 7.1.1-SNAPSHOT
started 2 months ago

4 C	11 D	0 A	2 A
Bugs	Vulnerabilities	New Bugs	New Vulnerabilities

Code Smells

67d A	2.6k	2d A	58
Debt	Code Smells	New Debt	New Code Smells

started 7 years ago

Coverage

90.1%	18k	87.4%
Coverage	Unit Tests	Coverage on 5.3k New Lines to Cover

Trunk-based Development

Is a version control management practice where developers merge small, frequent updates to a core "trunk" or main branch

- Allows for CI/CD
- Facilitates code review
- Keeps the trunk "green"

Develop in small batches

Feature flags

Implement comprehensive automated testing

Perform asynchronous code reviews

Have three or fewer active branches in the application's code repository

Merge branches to the trunk at least once a day

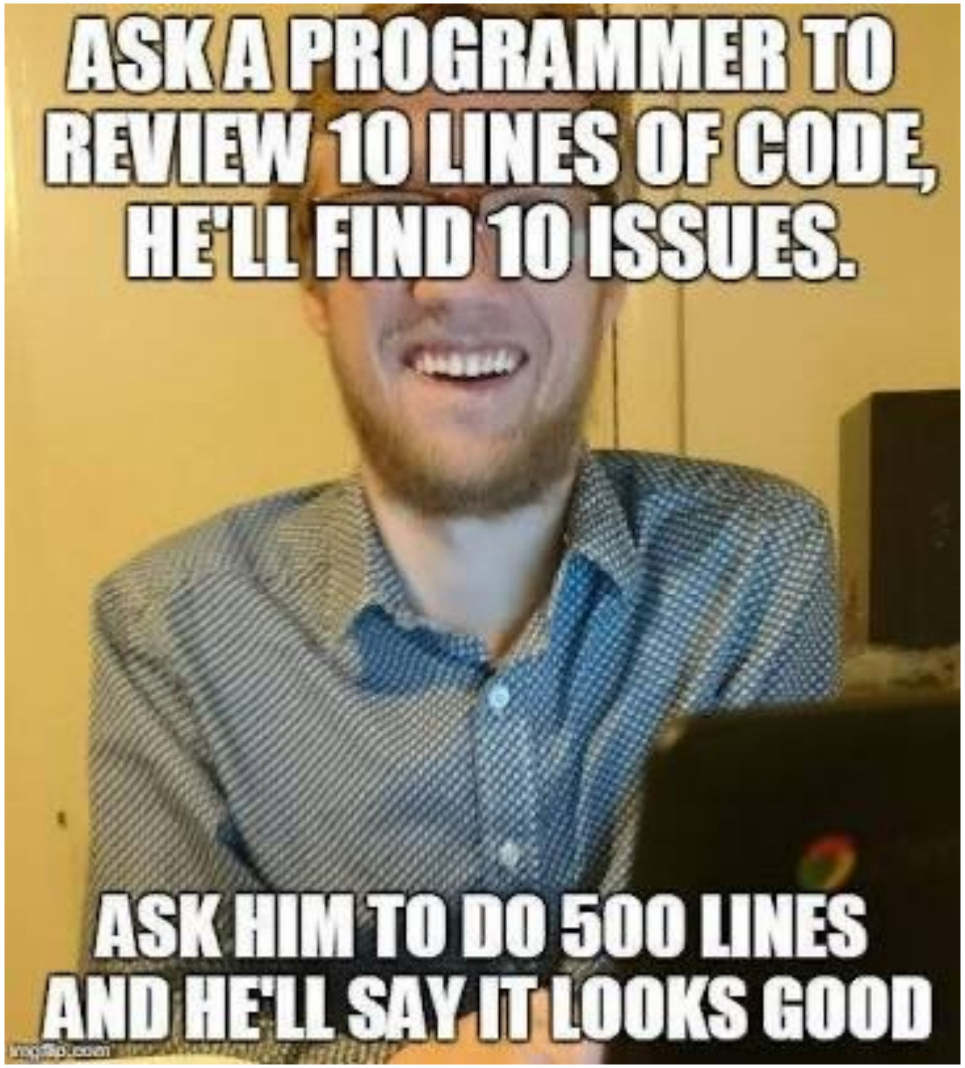
https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development

Google guide: What to look for in Code Review

https://google.github.io/eng-practices/review/reviewer/looking-for.html

Google guide: What to look for in Code Review

Functionality
Complexity
Tests
Naming & comments
Style & Consistency
Documentation
Every Line


ASK A PROGRAMMER TO REVIEW 10 LINES OF CODE, HE'LL FIND 10 ISSUES.
ASK HIM TO DO 500 LINES AND HE'LL SAY IT LOOKS GOOD
imgflip.com

https://google.github.io/eng-practices/review/reviewer/looking-for.html

Don’t accept CLs that degrade the code health of the system.

Most systems become complex through many small changes that add up, so it’s important to prevent even small complexities in new changes.

https://google.github.io/eng-practices/review/reviewer/looking-for.html

In doing a code review, you should make sure that:

- The code is well-designed.
- The functionality is good for the users of the code.
- Any UI changes are sensible and look good.
- Any parallel programming is done safely.
- The code isn't more complex than it needs to be.
- The developer isn't implementing things they *might* need in the future but don't know they need now.
- Code has appropriate unit tests.
- Tests are well-designed.
- The developer used clear names for everything.
- Comments are clear and useful, and mostly explain *why* instead of *what*.
- Code is appropriately documented (generally in g3doc).
- The code conforms to our style guides.

https://google.github.io/eng-practices/review/reviewer/looking-for.html

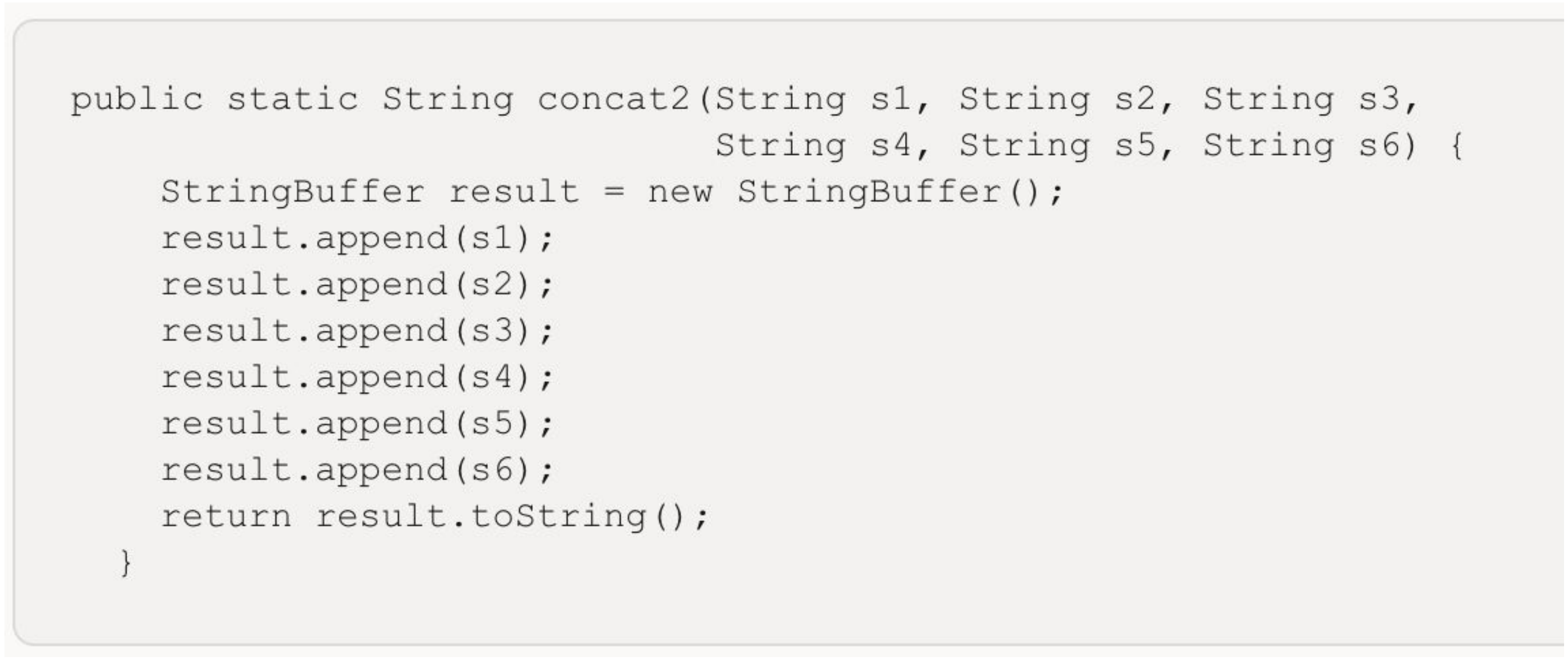
Complexity, modifiability, readability & over-engineering

“Too complex” usually means “can’t be understood quickly by code readers.” It can also mean “developers are likely to introduce bugs when they try to call or modify this code.”

A particular type of complexity is **over-engineering,** where developers have made the code more generic than it needs to be, or added functionality that isn’t presently needed by the system.

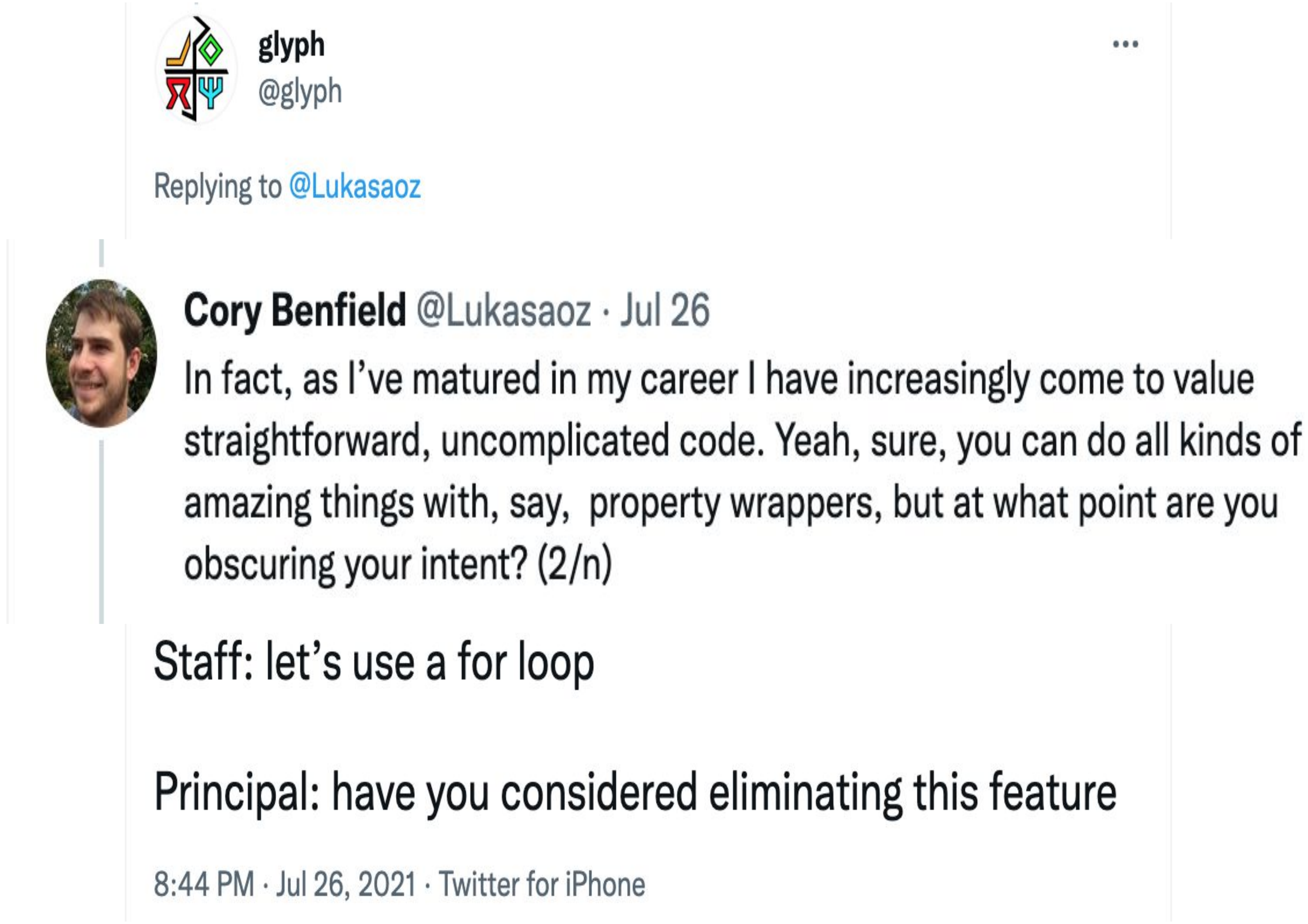
https://google.github.io/eng-practices/review/reviewer/looking-for.html

Senior developers write dumb code



```
public static String concat2(String s1, String s2, String s3,
                             String s4, String s5, String s6) {
    StringBuffer result = new StringBuffer();
    result.append(s1);
    result.append(s2);
    result.append(s3);
    result.append(s4);
    result.append(s5);
    result.append(s6);
    return result.toString();
  }
```

https://www.oracle.com/technical-resources/articles/javase/devinsight-1.html#1


glyph
@glyph

Replying to @Lukasaoz

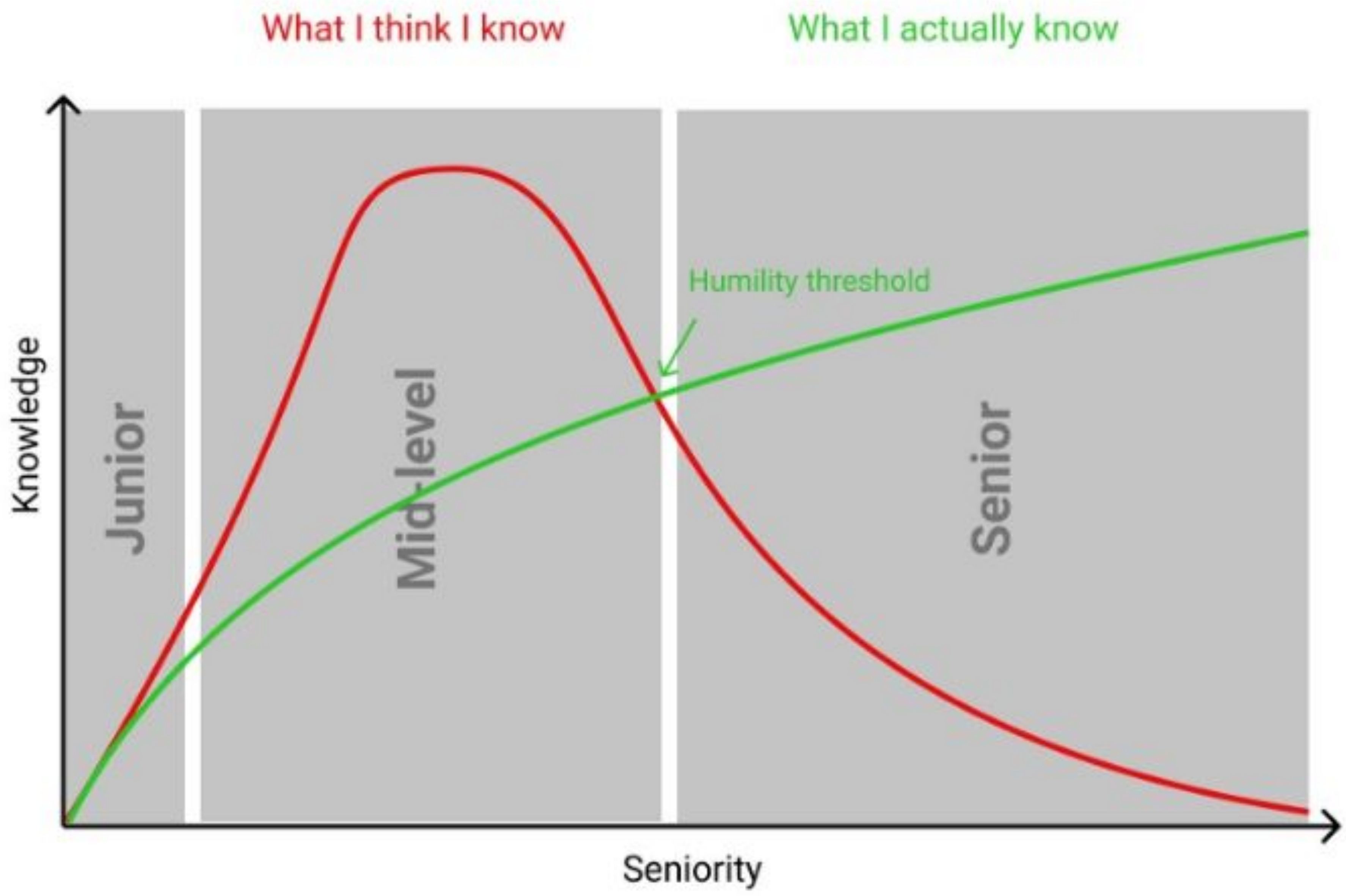
Cory Benfield @Lukasaoz · Jul 26
In fact, as I've matured in my career I have increasingly come to value straightforward, uncomplicated code. Yeah, sure, you can do all kinds of amazing things with, say, property wrappers, but at what point are you obscuring your intent? (2/n)

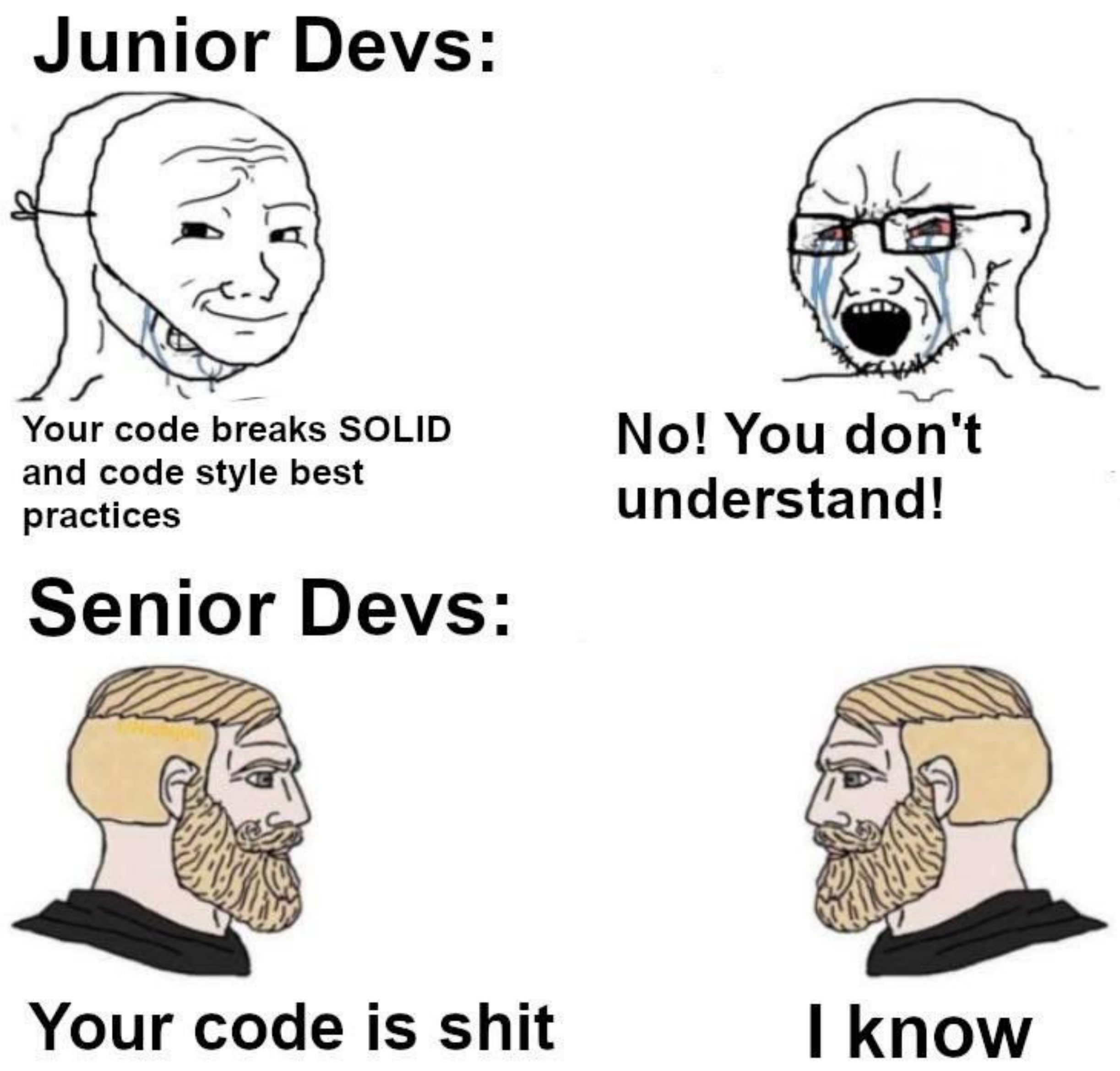
Staff: let's use a for loop

Principal: have you considered eliminating this feature

8:44 PM · Jul 26, 2021 · Twitter for iPhone

https://twitter.com/glyph/status/1419715153256325121


What I think I know
What I actually know
Humility threshold
Knowledge
Junior
Mid-level
Senior
Seniority


Junior Devs:
Your code breaks SOLID and code style best practices
No! You don't understand!
Senior Devs:
Your code is shit
I know

https://twitter.com/ElanShudnow/status/1438340230562603014

Functionality vs Changeability

First, developers must be able to *make progress* on their tasks. If you never submit an improvement to the codebase, then the codebase never improves.

On the other hand, it is the duty of the reviewer to make sure that each CL is of such a quality that the overall code health of their codebase is not decreasing as time goes on.

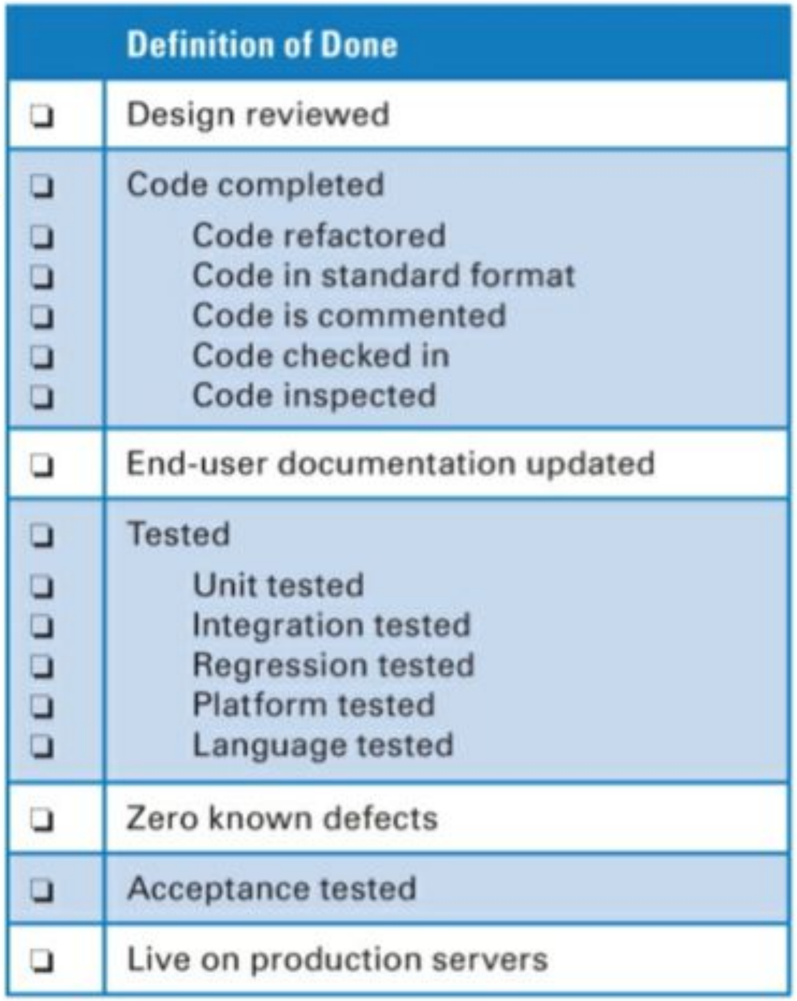
In general, reviewers should favor approving a CL once it is in a state where it definitely improves the overall code health of the system being worked on, even if the CL isn’t perfect.

Definition of Done

a checklist of the types of work…

…that the team is expected to successfully complete…

…before it can declare its work to be potentially shippable



	Definition of Done
☐	Design reviewed
☐	Code completed
☐	Code refactored
☐	Code in standard format
☐	Code is commented
☐	Code checked in
☐	Code inspected
☐	End-user documentation updated
☐	Tested
☐	Unit tested
☐	Integration tested
☐	Regression tested
☐	Platform tested
☐	Language tested
☐	Zero known defects
☐	Acceptance tested
☐	Live on production servers

Conclusion

- Start swift with “cheap” best practices.
- Validate that functionality is needed as quickly as possible.
- As time passes adopt more expensive best practices to be able to move quicker.
- When adopting practices: make sure you understand their essence
- Proceed gradually: decide critical things first
- Balance user perspectives vs engineering perspective