CHARACTERIZATION TESTING: MITIGATING RISK OF CHANGES AROUND LEGACY CODE.

Characterization Testing – written and presented by Martin Mayer

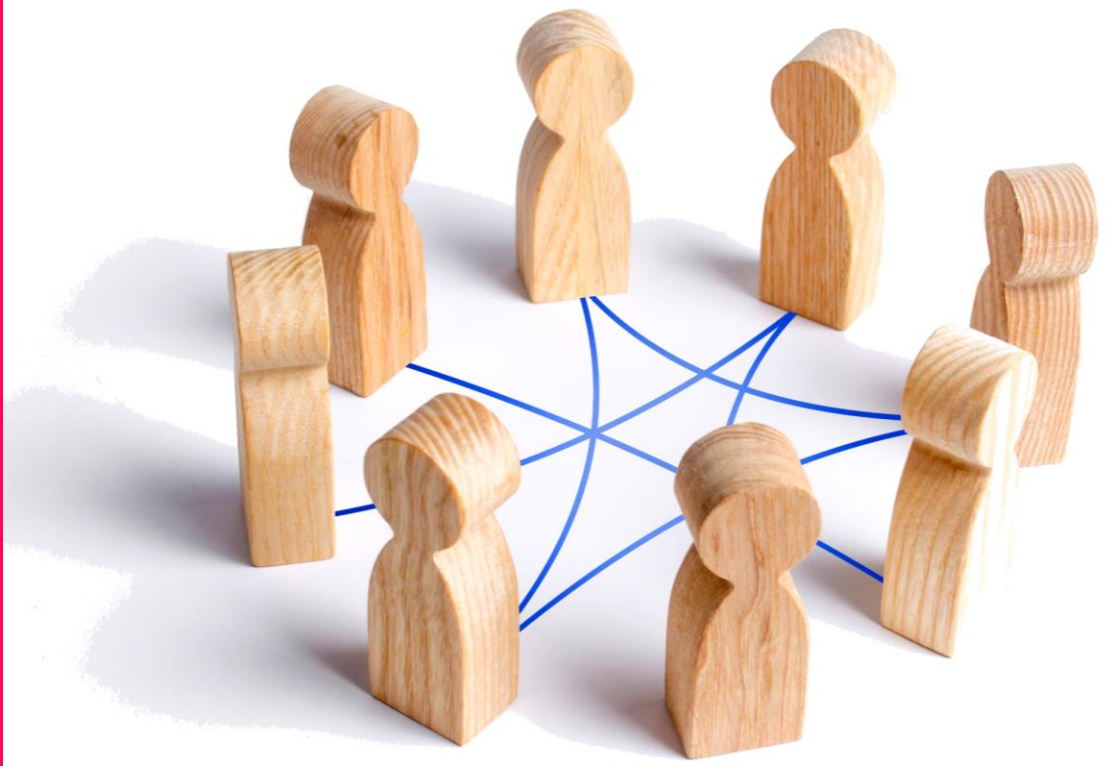
28.01.2021 The SocialCode.

Sorted.

MY APPROACH, NOT THE APPROACH.

Please contribute to add value to this session:

- Ask questions
- Give honest feedback
- Share your own experiences and conclusions



SCAN ME!

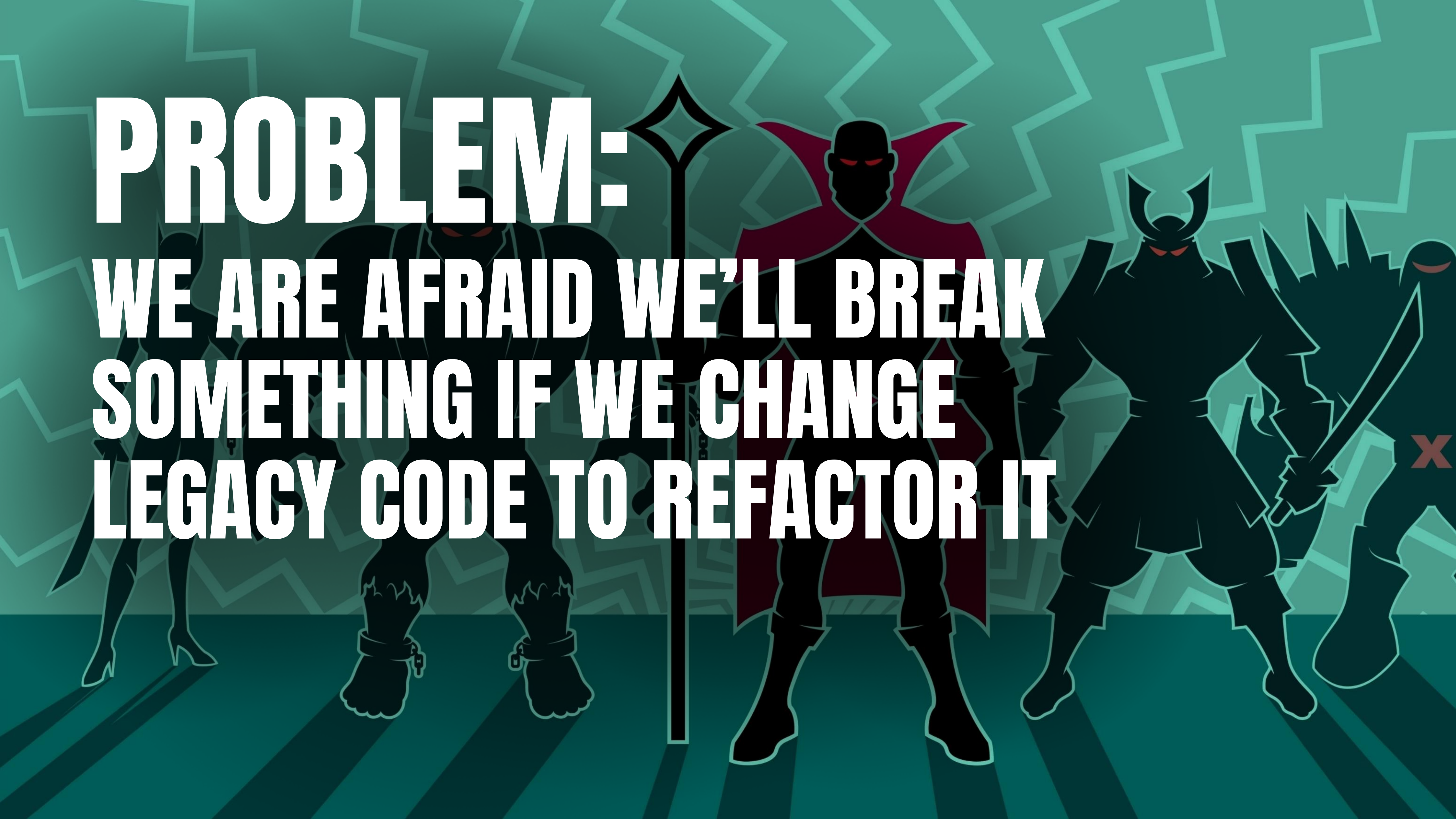
GLOSSARY OF TERMS, SLIDE DECK AND REFERENCES

http://bit.ly/39g9JNG

LEGACY CODE

CHANGE

TESTING



PROBLEM:

WE ARE AFRAID WE'LL BREAK SOMETHING IF WE CHANGE LEGACY CODE TO REFACTOR IT

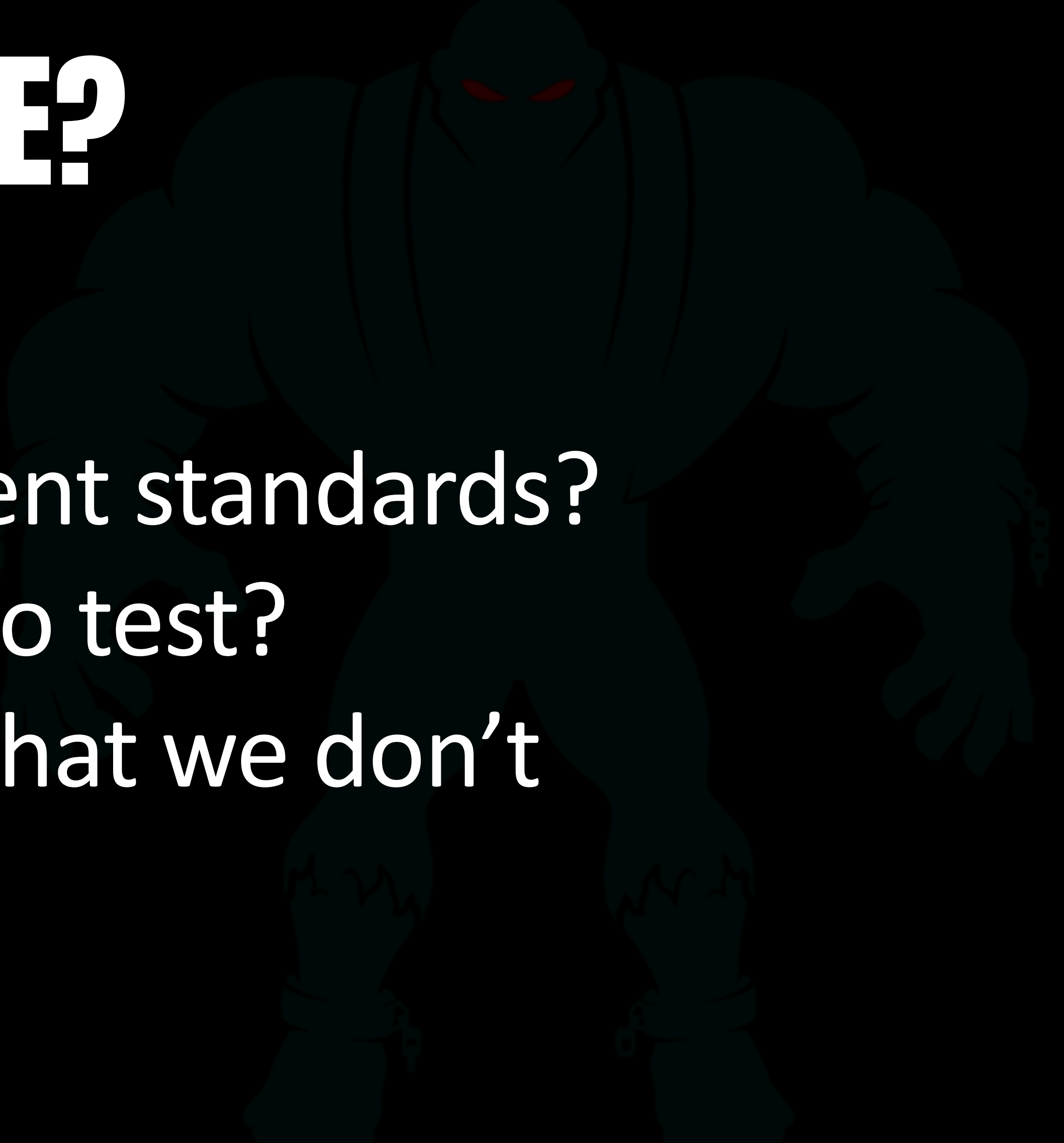
WHAT IS LEGACY CODE?

Old code?

Code written before current standards?

Bad code that is difficult to test?

Difficult-to-change code that we don’t understand?



WHAT IS LEGACY CODE?

“simply code without tests”

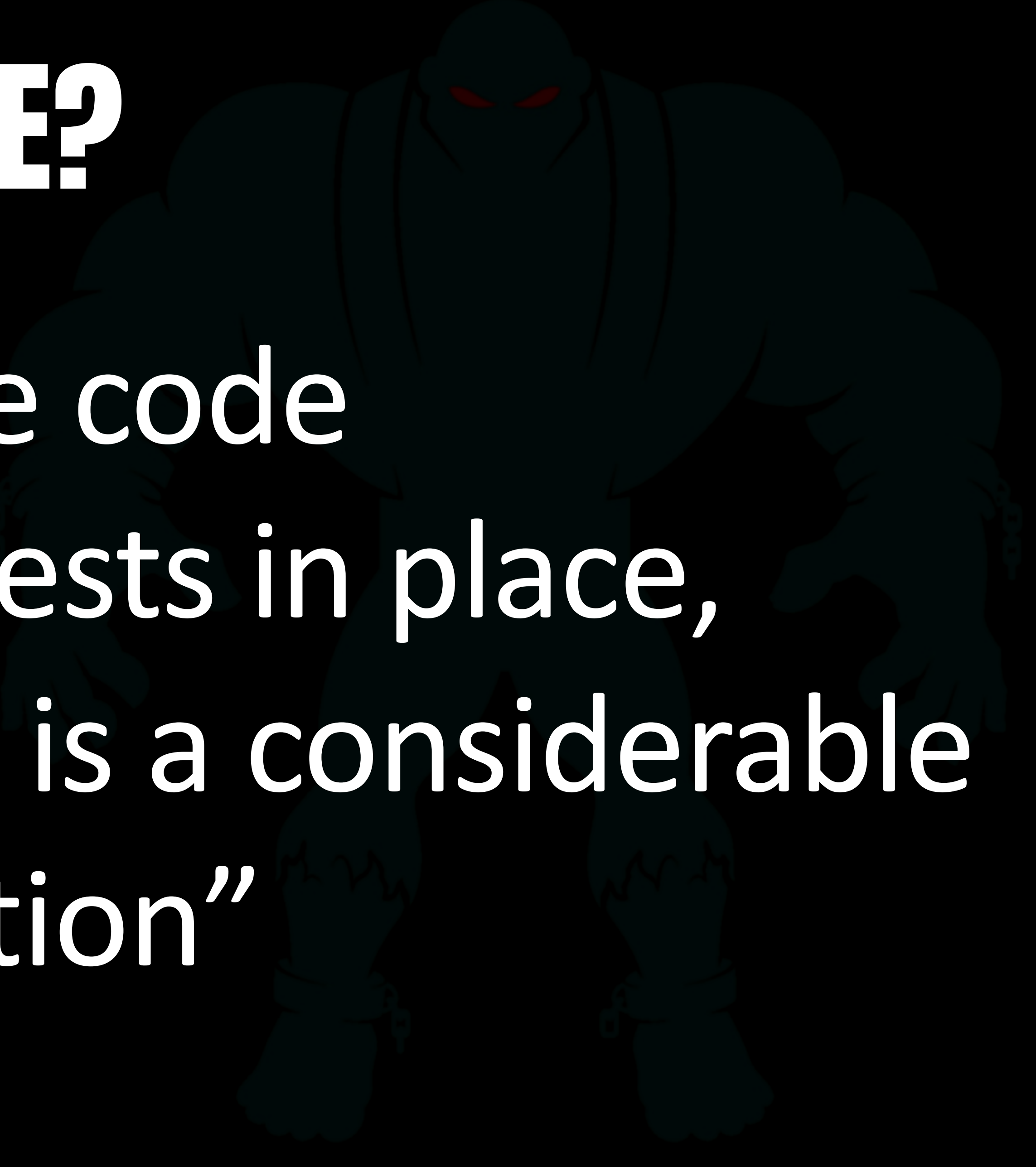
Working Effectively with Legacy Code, Michael Feathers

WHAT IS LEGACY CODE?

“profitable code
that we feel afraid to change”

Surviving Legacy Code with Golden Master and Sampling,
J. B. Rainsberger

WHAT IS LEGACY CODE?



“profitable/valuable code without adequate tests in place, meaning its change is a considerable risk to the organisation”

IF IT’S SO RISKY, THEN WHY CHANGE LEGACY CODE TO REFACTOR IT?

WHY CHANGE VALUABLE WORKING CODE?

Customer requirements

Bug fixes

Compliance

Security concerns

etc.

WHY REFACTOR?

Prepare for a value-adding change
Code may be reusable
Write code that is easier to read and debug
Make code changes safer and faster
Easier to estimate effort
Reduce time to skill up

POTENTIAL SOLUTION: CHARACTERIZATION TESTING

WHAT IS CHARACTERIZATION TESTING?

A form of exploratory testing

Produces (mostly) automated tests

Informs our regression cases

Forms a benchmark

Aims to capture and preserve all the value

REASONS FOR CHARACTERIZATION TESTING

1. Code isn’t well covered by tests
2. Requirements are unclear
3. Code isn’t clean and readable
4. Adding coverage would add safety

START TESTING!

START TESTING!

1. Start with the simplest unit of code
2. Assess if more logging should be added
3. Consider the structure of our test
4. Record all of the valued output to file
5. Create the test case source – our inputs

START TESTING!

6. “Arrange” – set up the test prerequisites
7. “Act” – call the subject under test
8. “Assert” – manual and automated options to assess if the result is “Pass” or “Fail”
   – We are focussed on the “diff” between the output of test runs: **before** and **after**

PROS

We can test code we don’t understand
Set up a benchmark before changes
Enables testing at relative speed
Easy general approach to learn and teach others
Supports open-box and closed-box testing
Fun to work with

CONS

Takes time and hard work

Can easily mean a lot of test cases and slow cycles

Depending on framework, a lot boilerplate code

Tests can be brittle as they depend on the detail

ALTERNATIVES / SUPPLEMENTARY TASKS

ALTERNATIVES / SUPPLEMENTARY TASKS: CODE AND REQUIREMENTS ANALYSIS

Understand the requirements and cover all code with appropriate unit, integration and end-to-end tests

ALTERNATIVES / SUPPLEMENTARY TASKS: FEATURE FLAGS

Set up a new version of the code (starting with a direct copy) and make modifications there

Preserve the original version in source using a “feature flag” stored somewhere in remote configuration to switch between versions

ALTERNATIVES / SUPPLEMENTARY TASKS: FEATURE BRANCHES

In our version control, branch off our main source to a new feature branch and make changes there – we can then deploy either that branch or main to a test environment and verify there – comparing results from the 2 deployments

ALTERNATIVES / SUPPLEMENTARY TASKS: MANUAL EXPLORATORY TESTING

Slow but can be used to research the code behaviour before writing characterization tests

Also a good alternative for components that are hard to test with automation

ALTERNATIVES / SUPPLEMENTARY TASKS: DEBUG EXISTING CODE

Debug to reverse engineer how the code is currently behaving

ALTERNATIVES / SUPPLEMENTARY TASKS: OBSERVABILITY

Develop additional logging to:

- trace behaviour
- monitor errors

ALTERNATIVES / SUPPLEMENTARY TASKS: AUGMENT AUTOMATED REGRESSION TESTS

Add to existing regression tests and introduce tests for newly refactored components as they are broken out

IMPLEMENTATION OPTIONS: OWN SIMPLE TEST FRAMEWORK

Write a test framework from scratch

Record outputs to file and play inputs from file

Capture logged string data to save as part of the output file

Compare the run to a pre-recorded output log in “working” state – manually or with an assertion

IMPLEMENTATION OPTIONS: APPROVAL TESTS LIBRARY

ApprovalTests library supports multiple languages

It standardises syntax but the test code is quite brittle and contains a lot of boilerplate

- I haven’t spent a lot of time with this yet

HOW CAN I TRY THIS?

- Follow the references listed in this slide deck
- Organise a session with your peers – at work or in a community like this
- Follow the tutorials with frameworks such as ApprovalTests

SCAN ME!

GLOSSARY OF TERMS, SLIDE DECK AND REFERENCES

http://bit.ly/39g9JNG

CONNECT

Connect with me on LinkedIn: https://bit.ly/3sYslJF

Linked in

ANY QUESTIONS?