Improving code quality

Kim Brugger

About me

Professional code monkey & jack of all trades

Master of: C, perl, python

DRD4-7R carrier

Blackbelt in puzzle solving

Not a finisher

Selective agile

Hate writing papers

The bitterness of poor quality remains long after the sweetness of meeting the schedule has been forgotten.

-Karl Wiegers

The plan for today

Introduction

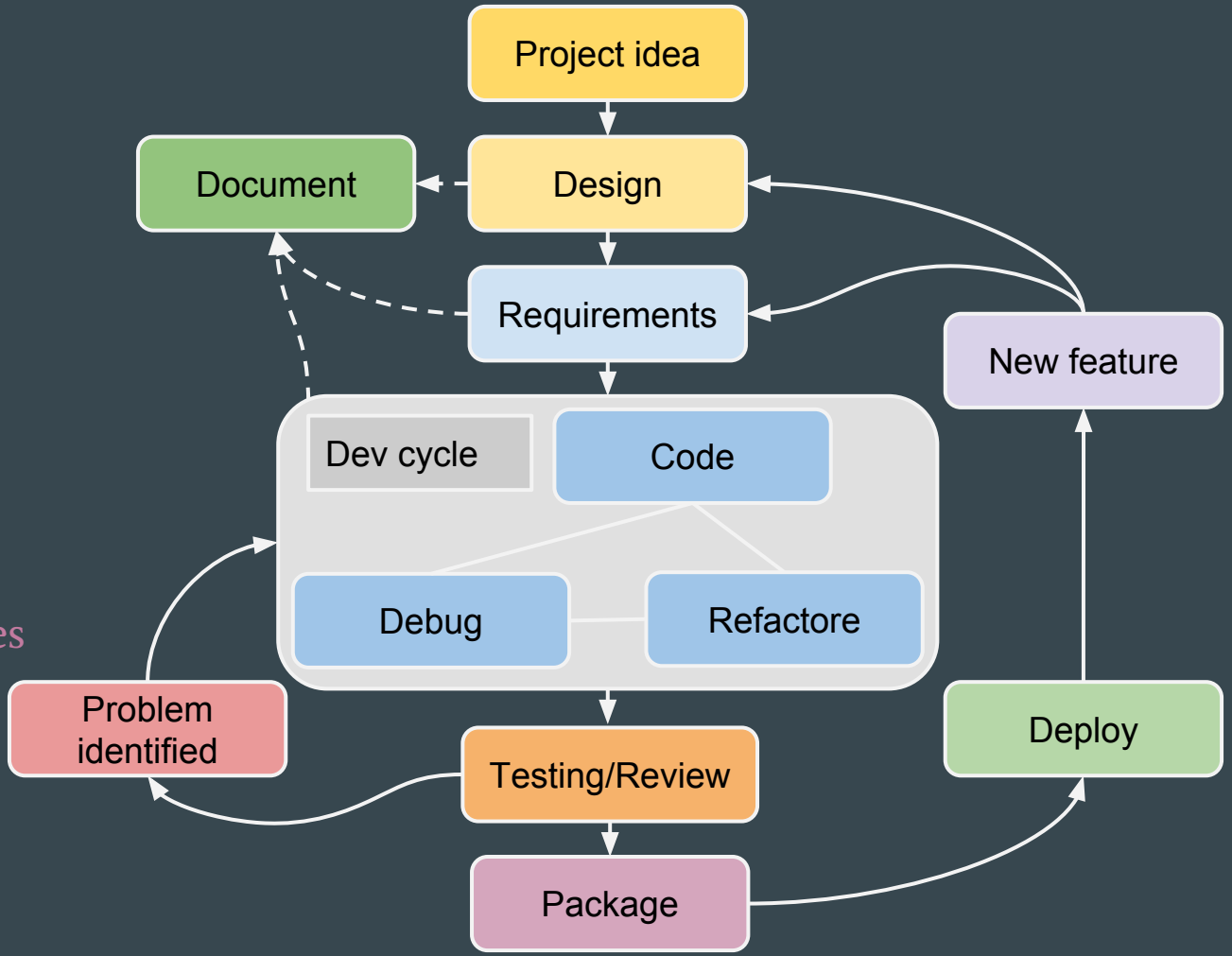
Debugging

Unit testing

Refactoring

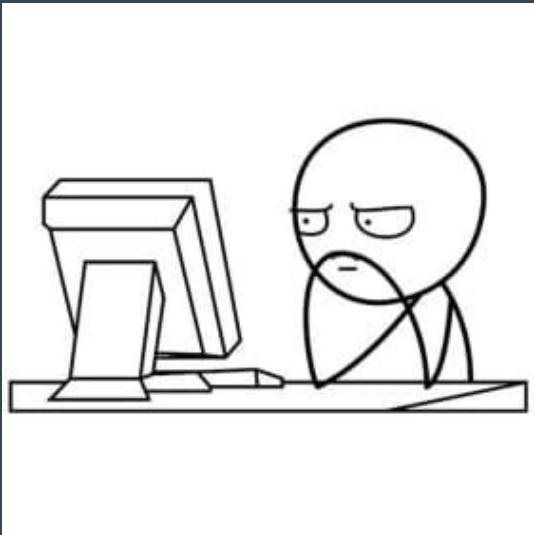
Packaging/delivery/releases

DevOps (CI/CD)



Introduction: It this you?

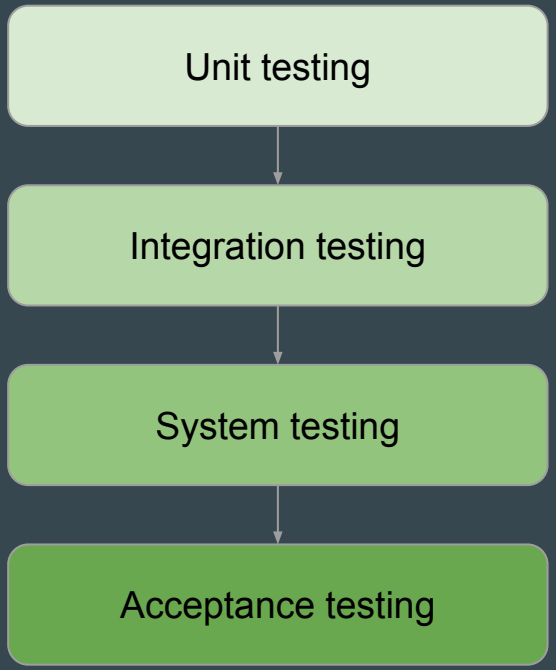
- I work best on my own
- I spend a lot of time debugging, and fixing bugs.
- My code is self explanatory, so doesn't need comments or other documentation.
- Code reviews are a waste of time as I am a superior programmer
- I don’t refactor nor write tests as I am too busy debugging.
- It works in my machine
- If they cannot get my software running, it is because they are st**id



Introduction: Hard earned wisdom

- Programming is an ever evolving skill
- Find your style
- Keep things simple and test continuously
- Don’t trust anything!
- Don’t be clever
- Know when to break the rules
- Adapting best practices is hard work

Introduction: the many layers of testing



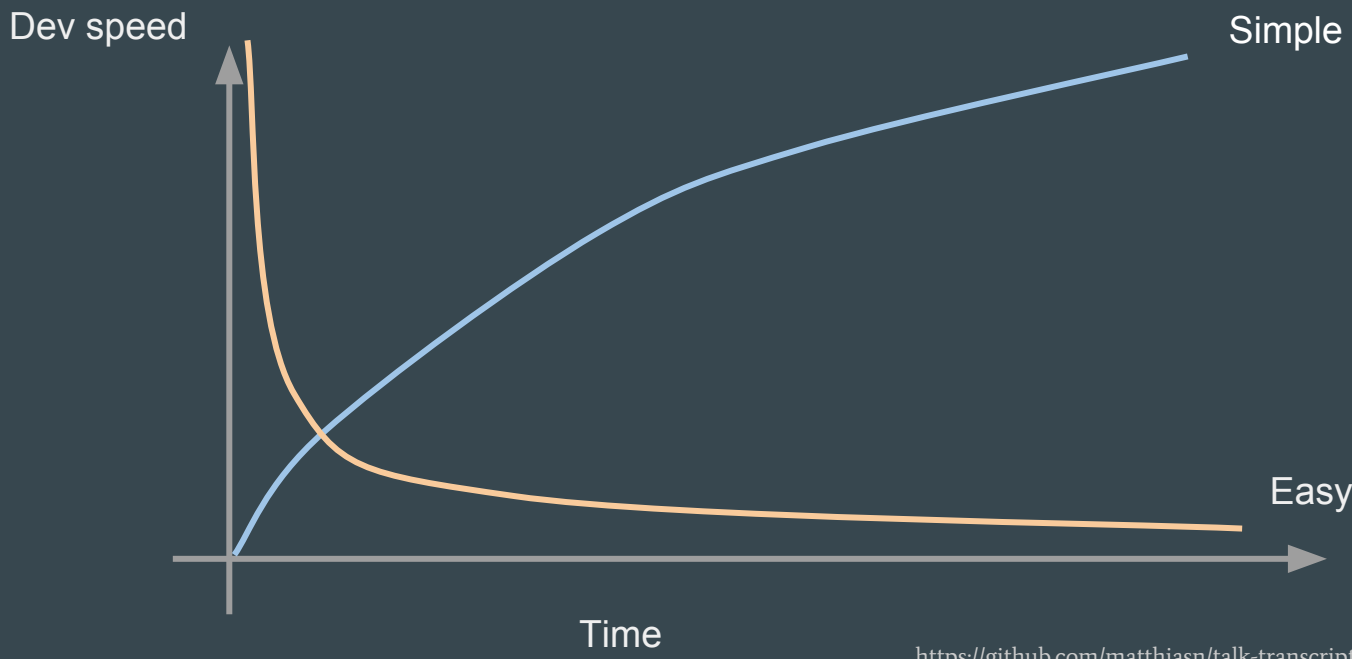
A level of the software testing process where individual units of a software are tested. The purpose is to validate that each unit of the software performs as designed. (whiteboxing)

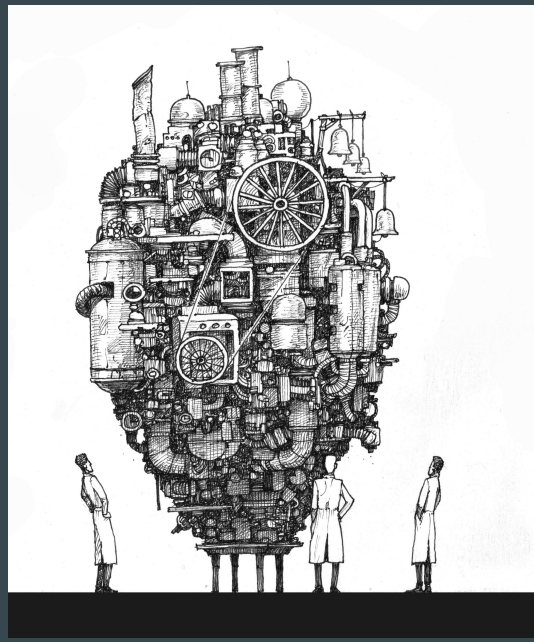
A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

A level of the software testing process where a complete, integrated system is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.

A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

Introduction: Simple vs Easy

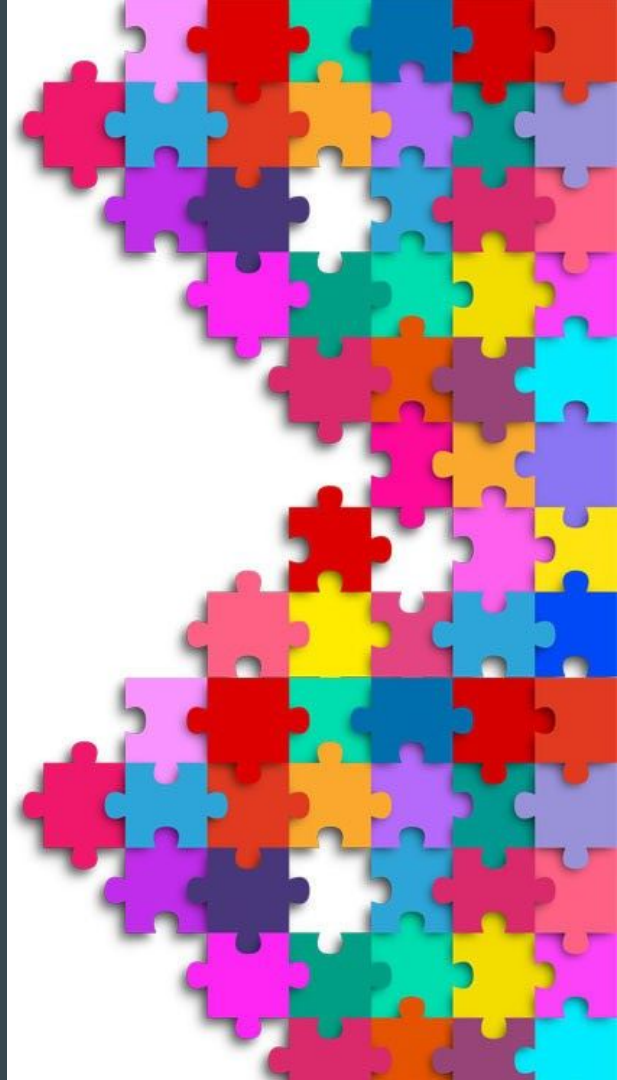




https://github.com/matthiasn/talk-transcripts/blob/master/Hickey_Rich/SimpleMadeEasy.md

Introduction: Modular design patterns

- Modularity and isolation of functionality
- Construct your program from parts:
  - Functions
  - Modules
  - packages/libraries
- Purity: functions should have no notion of state:
  - They take input values and return values
- Break your code down to more functions
  - if you have too many levels of indentation
  - if a function gets too long
  - if a function does more than one thing
  - if you find it hard to name a function



Introduction: Stand on the shoulder of giants

Dependent on others for features.

Impossible to fix errors and install

Build tools your specifications!

No peer review of design and code

Bigger chance of having undetected bugs

Introduction: Signs that things are not well

- Bugs start appearing in unexpected places when “fixing” unrelated issues
- You have no overview of flow of the code
- Small changes and features are hard to implement
- Quick hacks trumps design
- Messy codebase with unused functions, and functions in different versions
- A high reliance of global variables
- The complexity of the code hinders further development
- You have given up on keeping the documentation up to date
- You are in massive technical debt

Introduction: Today we will need

python

pytest

virtualenv

github account

https://github.com/cssd2019

Introduction: reflection & thoughts

How can I change my current practices?

Do you work alone or in a group?

Is a rewrite easier than fixing the errors?

How do you deal with legacy code

Various resources

https://coderefinery.org/

https://software-carpentry.org/lessons/

https://www.martinfowler.com/

https://docs.pytest.org/en/latest/

https://www.bioconductor.org/developers/package-guidelines/

https://python-packaging.readthedocs.io