http://bit.ly/ase-coderefinery-2019

Fail Forward: Development of Reproducible and Reusable software is a learning experience

Bjørn Lindi and Radovan Bast

NeIC/ NTNU & NeIC/ UiT The Arctic University of Norway

Credits: Jonas Juselius, Roberto Di Remigio, Ole Martin Bjørndalen

Fail forward?

Fail forward?

Give yourself the freedom to make mistakes, establish short feed back cycles.

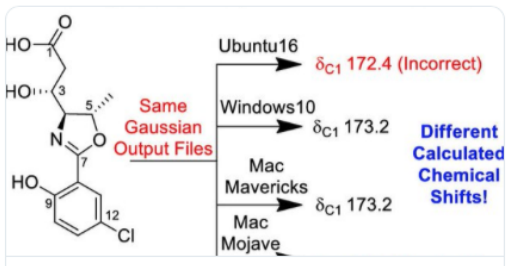
Reproducibility?

Reproducibility?

Doubt about 150 published chemistry studies

Lucas Moore @LucasCMoore · Oct 8, 2019

Holy crap. Huge bug uncovered in computational chemistry software because different operating systems sort files differently and the published scripts don't handle it well. If you do or rely on calculated NMR chemical shifts, this is a must-read.
pubs.acs.org/doi/10.1021/ac…



Characterization of Leptazolines A–D, Polar Oxazolines …
The bioactivity-guided examination of a Leptolyngbya sp. led to the isolation of leptazolines A–D (1–4), from the culture
pubs.acs.org

Patrick Willoughby
@pat_willoughby

Really great find by Rui and Prof. Williams. When I wrote the scripts 6 years ago, the OS was able to handle the sorting. Rui and Williams added the necessary sort code and added a function to ensure the calcs were properly aligned. Kudos!

♡ 44 9:58 PM - Oct 8, 2019

15 people are talking about this

Reusable?

Reusable?

The bush versus the tree

Write the tests first!

Write the tests first!

.. but I know what I am doing, why should I write the tests first?

What you "know" is a mix of...

- True knowledge
- A set of assumptions

What you "know" is a mix of...

- True knowledge
- A set of assumptions

Some of the assumptions are wrong.

A test written first can reveal this.

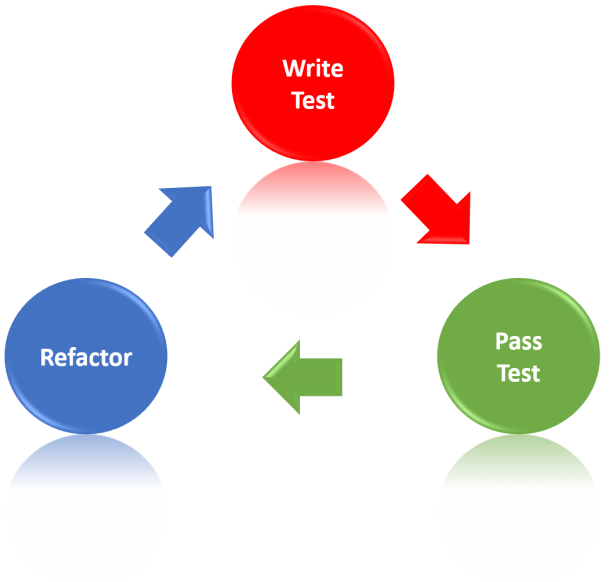
Implementing a feedback cycle

- By writing tests first you implement a feedback cycle.
- Writing tests specifying behavior, you create a active environment which relates to your code.
- If behavior is broken, it shows up immediately.

Test-first development

- It is a design methodology.
- "It helps developers build high quality code by forcing them to write testable code and by concretizing requirements." --David Scott Bernstein [1]

Learn Test-Driven Development



Starting on red.

- Focus on one test at a time, and implement the new behavior step by step - with short feedback cycles.
- Name the test properly - it is the test of a new behavior/feature.

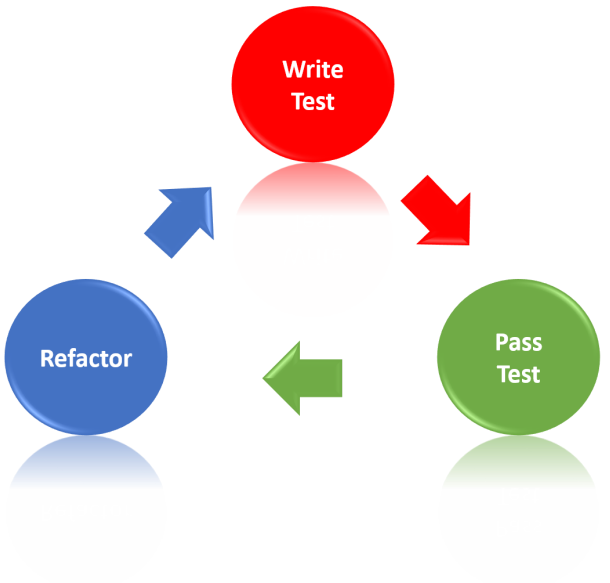
Getting to green.

- We only write as much code as needed to pass the test. If implies copy code, we copy code. If mean using constants, we use constants.
- First we solve "that works" part of the problem. Then we solve the "clean code" part(that is part of the next step - refactoring). Divide et imperia.

Refactor - incorporate the learning experience from satisfying the test

- To get passed the test, you did some sins. Now you make it right.
- Get rid of duplication.
- Let the code express your learning from the outside-in/inside out ping-pong.
- Make the code readable and understandable.
- Tidy up and make the code CLEAN.

Start over again, add new functionality



What is CLEAN code?

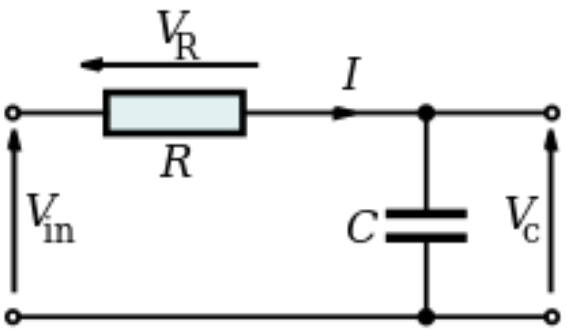
What is CLEAN code?

- It is Cohesive.
- It is Loosely coupled.
- It is Encapsulated.
- It is Assertive.
- It is Non-redundant

This is taken from [1].

Quality Code is Cohesive

- In software development cohesive means entities should have a single responsibility.



Quality Code is Loosely Coupled

- "Code that is loosely coupled indirectly depends on the code it uses so it is easier to isolate, verify, reuse and extend." [1]

Slide taken from [Complexity in software development by Jonas Juselius]

Quality Code is Encapsulated.

- Encapsulated code hide implementation details from the rest of the world.
- You separate what something does from how it is done, which gives you freedom to change how later on.

Quality Code is Assertive

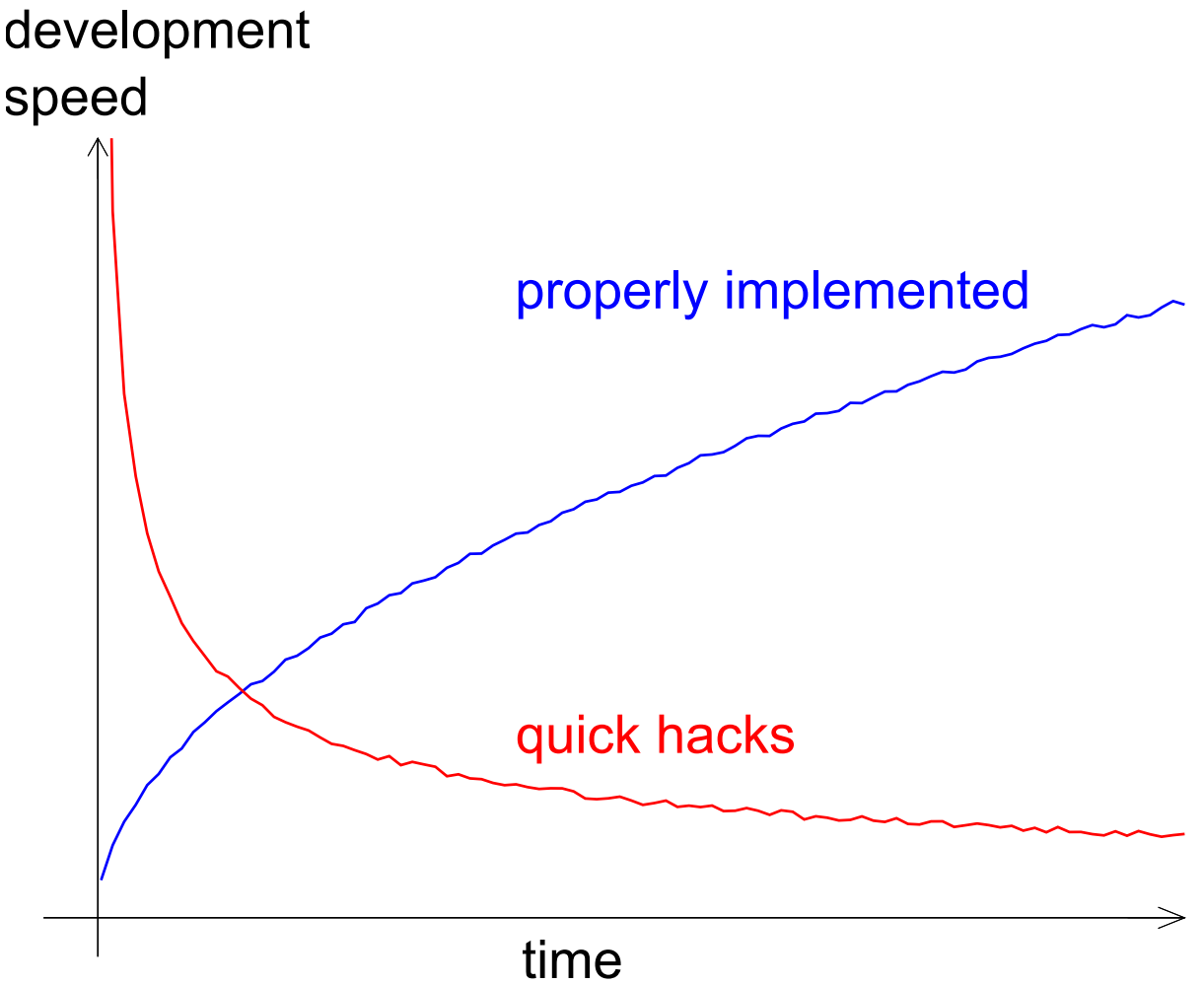
- The opposite is inquisitive: *Don't be so inquisitive. It's none of your business.*
- Software objects should not be inquisitive; they should be authoritative, in charge of them self.

Quality Code is Nonredundant

- Don't repeat your self (DRY)

That was a nice acronym - CLEAN - So what?

Increase Quality today to increase Velocity to tomorrow.



Version Control System

Central repository/ project place

"...packages has grown rather organically..."

"...packages has grown rather organically..."

- You are using a version control system like git.
- You have some experience with at least one aspect of CLEAN code.
- You have developed a sense of how to test and develop code incrementally (TDD)

"...packages has grown rather organically..."

- You are using a version control system like git.
- You have some experience with at least one aspect of CLEAN code.
- You have developed a sense of how to test and develop code incrementally (TDD)

This is necessary experience to be able to work with Legacy Code.

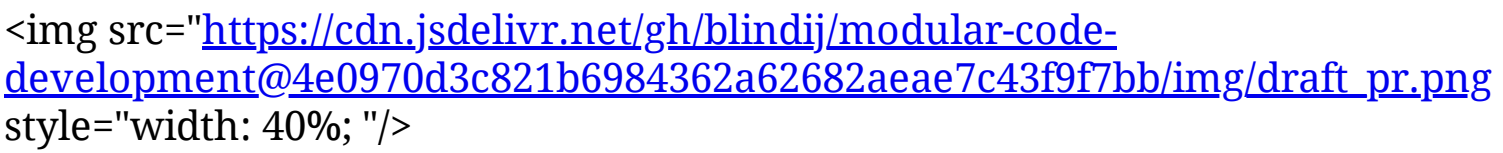
Other feedback cycles you should establish:

- Pair programming
- Code-review

Pair programming and Test Driven Development

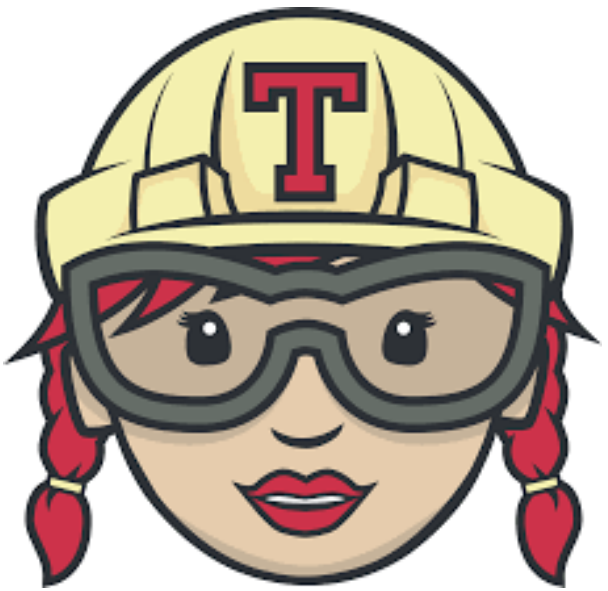
"TDD supported with pair programming is a natural fit. Learning TDD is made dramatically easier with a support system in place. Developers are more likely to revert to old, non-TDD habits without a bit of peer pressure from their teammates. Sitting with an experienced TDDer can be more than half the time need to ingrain the habit of TDD. Swapping pairs can help ensure that tests are written first and with care."

Code Review


<img src="https://cdn.jsdelivr.net/gh/blindij/modular-code-development@4e0970d3c821b6984362a62682aeae7c43f9f7bb/img/draft_pr.png
style="width: 40%; "/>

Automate testing and checking of code coverage

- Use a Continuous Integration service like Travis for automatic testing.
- Use a coverage service like Coveralls to verify coverage automatically.



Licensing

- Think about how you would like your software to be used and cited.
- Don't lock yourself out from using your code later.
- David Heinemeier Hansson about MIT License, Open Source [6]

References

- [1] Beyond Legacy Code - Nine practices to extend the life (and value of) of Your Software , by David Scott Bernstein
- [2] Test Driven Development: By Example, by Kent Beck
- [3] Modern C++ Programming with Test-Driven Development, by Jeff Langr
- [4] Pro Git, by Scott Chacon and Ben Straub
- [5] tmux 2 productive mouse-free development by Brian Hogan, The Pragmatic Programmers / Chapter 5 pair programming with tmux
- [6] "Open source beyond the market"
- Cicero:Serving presentation slides written in Markdown
- CodeRefinery workshops: https://coderefinery.org/workshops