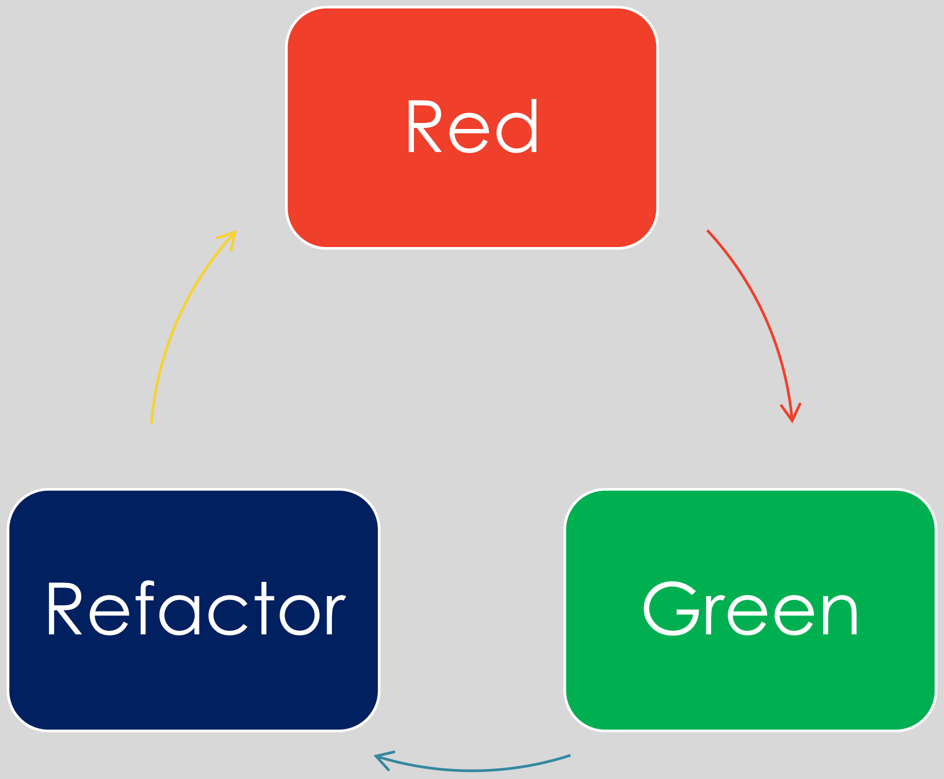
UNIT TESTING AND TEST-DRIVEN DEVELOPMENT

“If you don’t like unit testing your product, most likely your customers won’t like to test it either.”

TDD Cycle



Why TDD?

“

When doing TDD, the purpose of a test is to provide feedback about the API you're developing. A unit test is the first client of the production API. If a test is difficult to write, the production API is difficult to use.

- Mark Seeman

TDD encourages better code

- Leads to decoupled code
  - Naturally leads to following principles like:
    - *Single-responsibility principle*
    - *Interface segregation principle*
    - *Dependency inversion/injection*
- Code smells, poorly designed or difficult to use code become more apparent sooner in the design process

Conclusion

- Demonstration of TDD and unit testing
- Examples with C# and Javascript
- Provided a contrast between code that was written without tests in mind versus testable code
- Demonstrated how by writing tests first, testing and mocking became easier
- Some brief examples of C# 8 features that lead to more declarative (and therefore easier to test) code
    - Pattern Matching
- Example usage of the “result” type concept to reduce use of nulls and signal intent better
- Javascript and Vue example unit testing

Further reading

“What to test and what not to test”

https://blog.ploeh.dk/2018/11/12/what-to-test-and-not-to-test/

“Discerning and maintaining purity”

https://blog.ploeh.dk/2020/02/24/discerning-and-maintaining-purity/

“TDD test suites should run in 10 seconds or less“
https://blog.ploeh.dk/2012/05/24/TDDtestsuitesshouldrunin10secondsorless/

“Putting cyclomatic complexity to good use”

https://blog.ploeh.dk/2019/12/09/put-cyclomatic-complexity-to-good-use/

.NET Libraries shown: Xunit, Shouldly, AutoFixture, NSubtitute, Optional

JS libraries shown: Vue, Vue Test Utils, Jest