


Agile Engineering Practices

A Module in Agile Fundamentals




What We Will Cover

- Unit Testing and Test Driven Development
- Pairing
- Smells
- Refactoring
- Generalization and Specialization




Unit Testing



Unit Test


“A Unit Test is a procedure used to validate that individual units of functional code are working properly. “

- Is usually done by developers
- Improves quality
- Facilitates changes
- Simplifies integration
- Enables automation
- Provides effective system documentation
- Makes your life simpler



Unit Testing

The Agile Way: Test First



Test Driven Development

"Test-Driven Development (TDD) is an evolutionary approach to development which instructs you to have test-first development intent. Basically, you start by writing a test and then you code to elegantly fulfill the test requirements."

- Small successful, tested steps.
- Do the simplest thing that could possibly work.

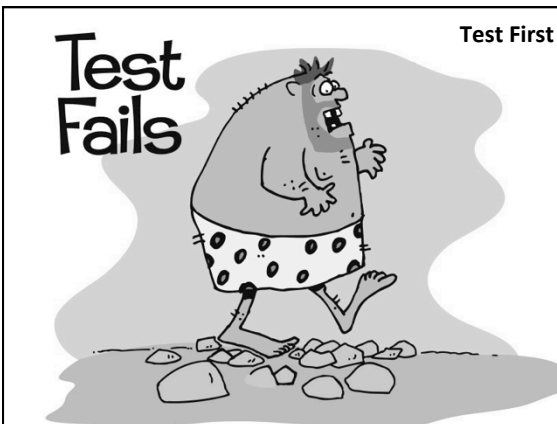


A User Story

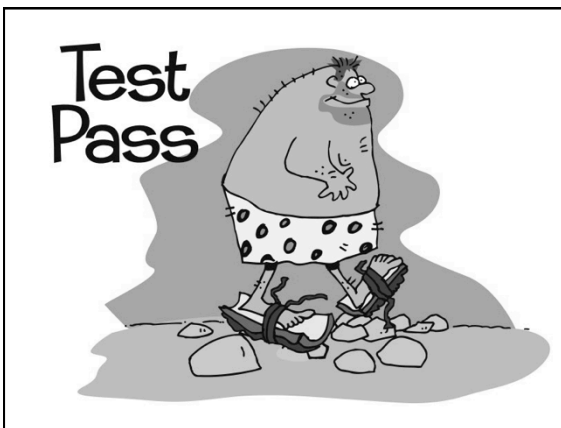


Test Fails

Test First









Uncle Bob Martin's Three Laws of TDD

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.



Pairing



Pairing

Why it sucks, and why you should hate it



It's distracting!

Hard to focus on the problem at hand
when your pair is sitting next to you
talking



Costly

Two people doing the work of one



Low morale

Only spend 50% of your time doing
any actual work



Hygiene

What?! I have to shower every day?!



Stop slowing me down!

Explaining thoughts to my pair will slow me down

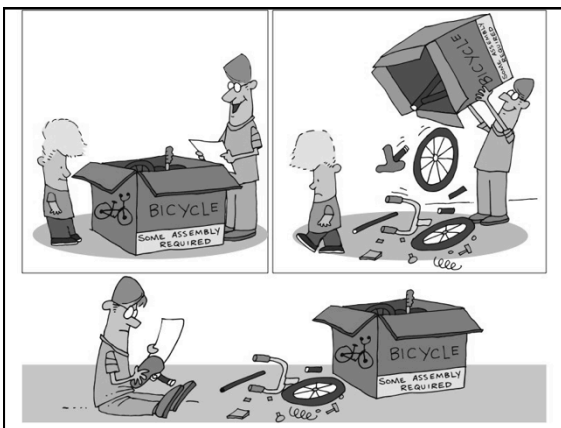


Pair Programming

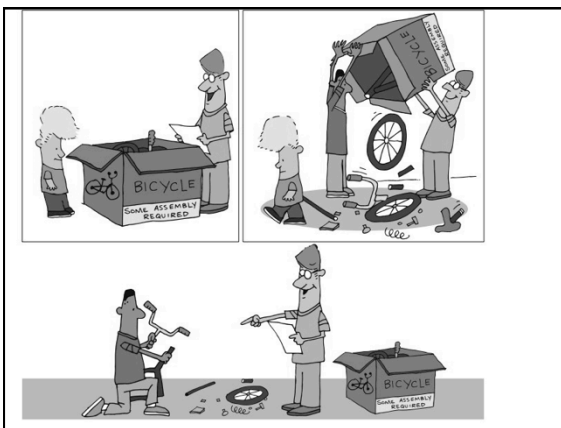
Now let's talk about the reasons you should do it

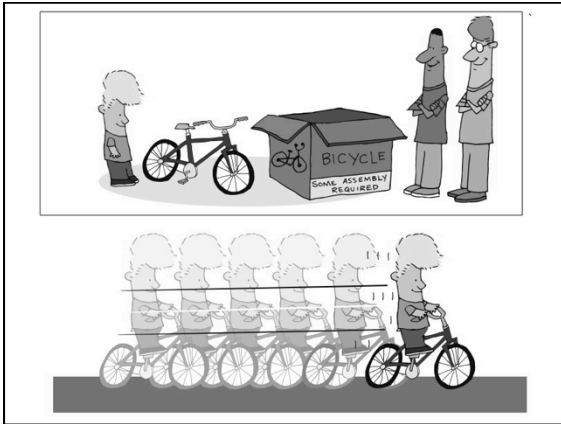


Development The Waterfall Way



The Agile Way: Pairing





Studies

- Cockburn, Alistair; Williams, Laurie (2000). "The Costs and Benefits of Pair Programming"
- Lui, Kim Man; Keith C. C. Chan (September 2006). "Pair programming productivity: Novice-novice vs. expert-expert"
- Arisholm, Erik; Hans Gallis, Tore Dybå, Dag I.K. Sjøberg (February 2007). "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise"
- Lui, Kim Man; Keith C. C. Chan, John Teofil Nosek (March/April 2008). "The Effect of Pairs in Program Design Tasks"
- Hannay, Jo E.; Tore Dybå, Erik Arisholm, Dag I.K. Sjøberg (July 2009). "The Effectiveness of Pair Programming: A Meta-Analysis"



Studies of Pair Programming

- Pair programmers are only 15% slower than two individuals but produces 15% fewer defects
- 100% agreed that they had more confidence in their solution than when they program alone
- 96% agreed that they enjoy their job more than when programming alone



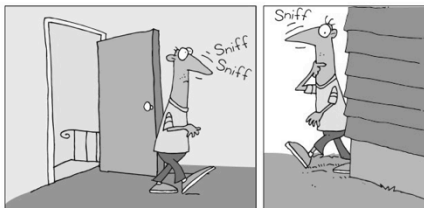
Pairing

- Find someone to pair with
- You are going to write a story
- One writes, one makes sure rules are followed
- Swap, and do it again



Understanding Smells

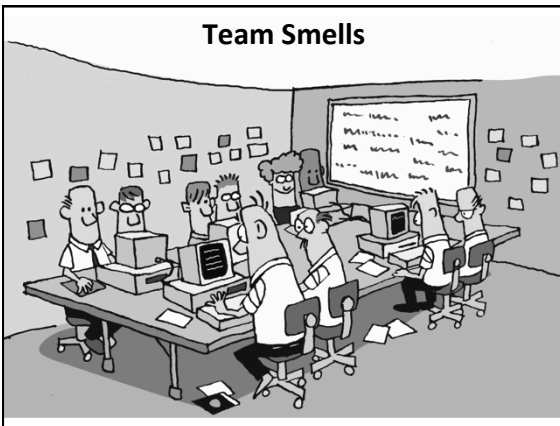




Pairing and Smells



Team Smells



Refactoring

ThoughtWorks
studios

Practically Speaking.. What is Refactoring?

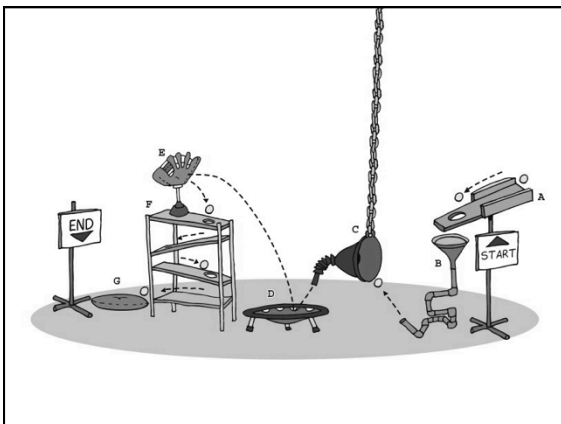
A series of *small* steps, each of which changes the program's internal structure without changing its external behavior

- Verify no change in external behavior by
 - Testing
 - Being very, **very** careful



The Need for Refactoring





Why Refactor

- To make room for new functionality
- To make the program easier to change
- To make the software easier to understand
- To "Fix broken windows"



When to/not to Refactor

Refactor when.....

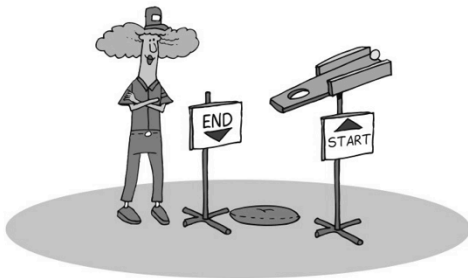
- Adding functionality
- Fixing a bug
- Doing a code review

Consider not refactoring when....

- Code is too messy (better to start over)
- Near a deadline



What about ownership?



Refactoring versus Redesign

- Refactoring - key part of modern software development practice –don't need permission
- Redesign - Large scale refactoring/redesign decisions should be owned by the whole team



Team practices

- Encourage refactoring culture
- Provide sound testing base
- Shared Code Ownership



Continuous Integration

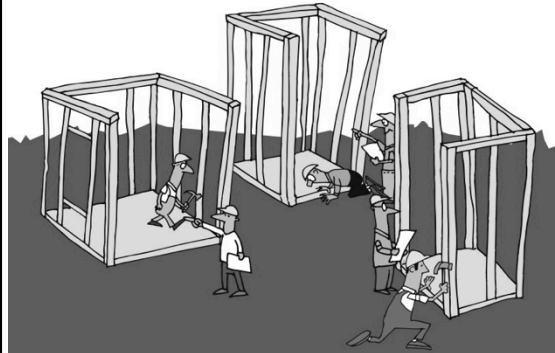


"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly."

<http://www.martinfowler.com/articles/continuousIntegration.html>



Independent Action



Check and Fail Early

