Pragmatic, Not Dogmatic TDD: Rethinking How We Test

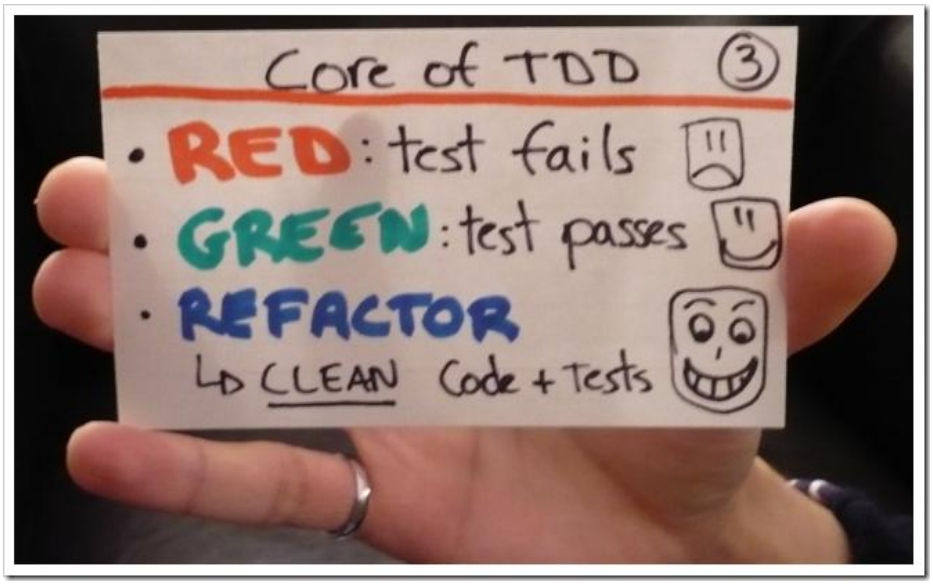
Rebecca Wirfs-Brock
&
Joseph W. Yoder

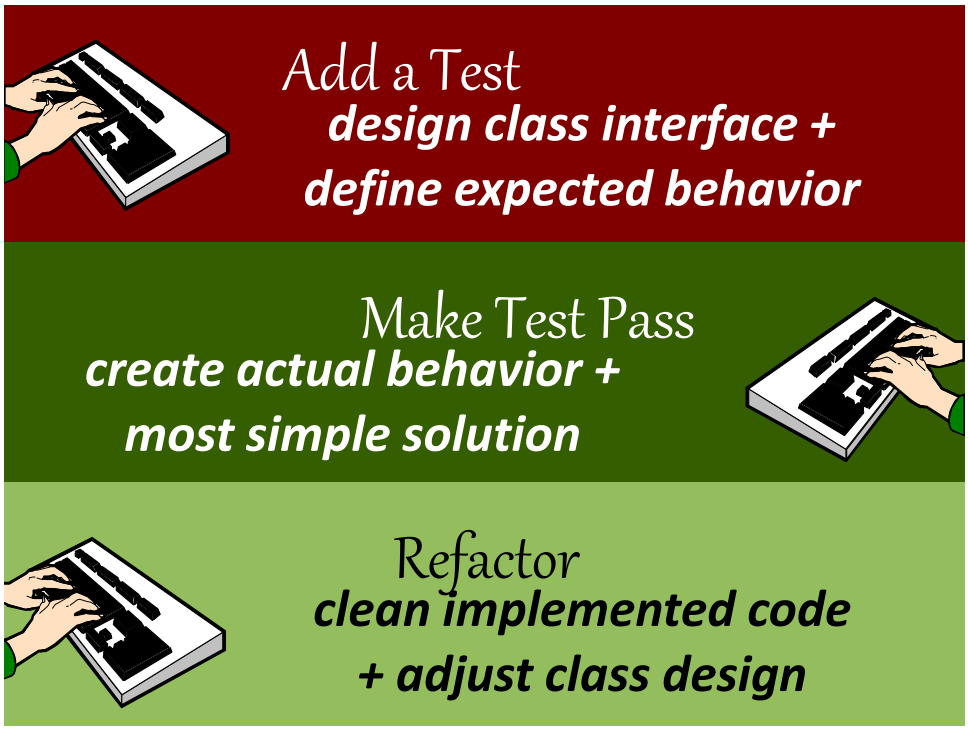
Introducing Joseph

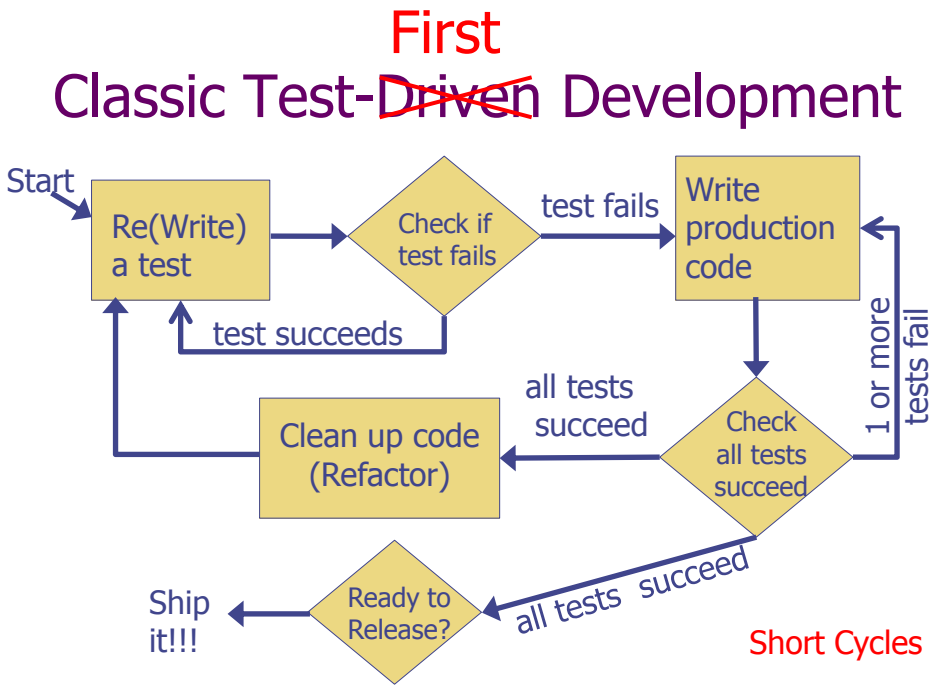
- Founder and Architect, The Refactory, Inc.
- Pattern enthusiast, author and Hillside Board President
- Author of the Big Ball of Mud Pattern
- Adaptive systems expert (programs adaptive software, consults on adaptive architectures, author of adaptive architecture patterns, metadata maven, website: adaptiveobjectmodel.com)
- Agile enthusiast and practitioner
- Business owner (leads a world class development company)
- Consults and trains top companies on design, refactoring, pragmatic testing
- Amateur photographer, motorcycle enthusiast, enjoys dancing samba!!!

Introducing Rebecca

- President, Wirfs-Brock Associates
- Agile enthusiast (involved with experience reports since 1st agile conference, board president Agile Open Northwest)
- Pattern enthusiast, author, and Hillside Board Treasurer
- Old design geek (author of 2 object design books, inventor of Responsibility-Driven Design, advocate of CRC cards, hot spot cards, & other low-tech design tools, IEEE Software design columnist)
- Consults and trains top companies on agile architecture, responsibility-driven design, enterprise app design, agile use cases, design storytelling, pragmatic testing
- Runs marathons!!! Slowly.




Add a Test
design class interface +
define expected behavior
Make Test Pass
create actual behavior +
most simple solution
Refactor
clean implemented code
+ adjust class design


First
Classic Test-Driven Development
Start
Re(Write) a test
Check if test fails
test fails
Write production code
test succeeds
1 or more tests fail
all tests succeed
Clean up code (Refactor)
Check all tests succeed
Ship it!!!
Ready to Release?
all tests succeed
Short Cycles

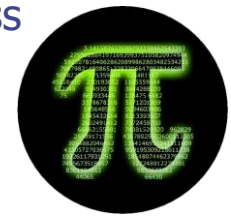
TDD Promises

1. Tests help you build the right thing
2. Tests guide development
3. Tests keep you focused
4. Tests allow you to change code safely and quickly
5. Tests ensure what you build works
6. You will end up with quality code
7. You will end up with a good design

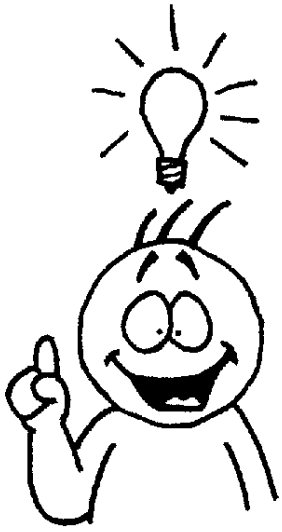
Question: Do tests help you build the right things?

Common Misperceptions

- Tests verify program correctness
  - 3.1415926535…
- Tests enable and encourage well-designed code



Understanding Tests

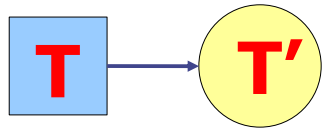


T **Test Target** → the thing we are trying to test.

Action → changes environment or the Test Target.

Assertion → compares expected vs observable outcome of the action on the Test Target.

Test → a sequence of at least one action and one assertion



Good Test Outline

1. Set up
2. *Declare the expected results*
3. Exercise the test
4. Get the actual results
5. Assert that the actual results match the expected results
6. Teardown

Pragmatic Testing Questions

What is important **to test**?

Who should write them?

Who runs them?

When are they run?

How are they run?

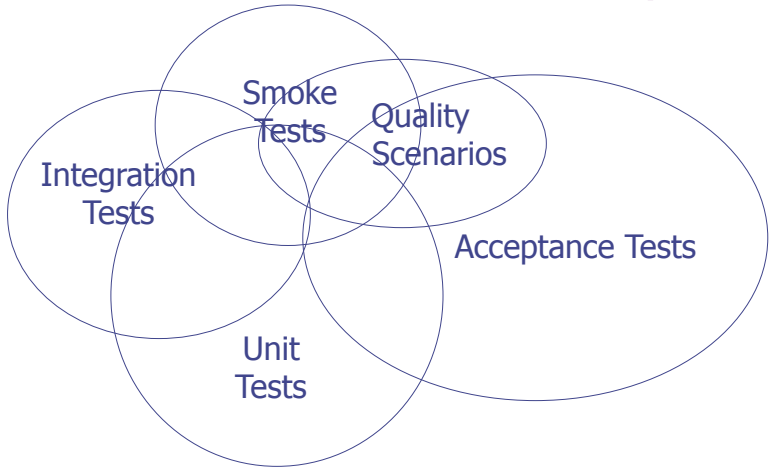
What to Test

- Significant scenarios of use, not isolated methods.
- The difficult parts: Complex interactions, intricate algorithms, tricky business logic
- Required system qualities
  - Performance, scalability, throughput, security...
- How services respond to normal and exceptional invocations.

What **Not** to Test

- Tests should add value, not just be an exercise.
- Do not test:
  - setters and getters
    (unless they have side effects or are very complex)
  - every boundary condition; only test those with significant business value
  - every exception; only those likely to occur or that will cause catastrophic problems.

Different Tests Can Overlap…



Common to only focus on Unit Tests in TDD!!!
We believe TDD is **more** than just Unit Tests

Question: Do tests help you build the right things?

Answer: Yes. But …

Question: Do tests guide development and keep you focused?

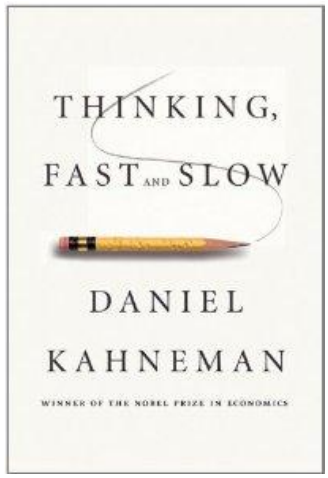
Two Faces of Testing

- Defining tests helps limit scope and increases focus
- Tests force you to implement functionality instead of jumping around and tweaking stuff

- But you may be missing the bigger picture…
- Might need to consider how current functionality affects the rest of the system

Thinking Fast vs. Slow

- Fast thinking: decisions based on intuition, biases, ingrained patterns, and emotions
- Slow thinking: Reasoning, logical thinking



Take Time For Both

- Slow thinking
  - Pairing and discussing options or why you want to implement something a certain way
  - Sketching, noodling, design spikes
- Fast thinking
  - Following your intuition, deciding on the fly
  - Fast turns of coding, testing and quick fixes… (Red/Green)

Question: Do tests guide development and keep you focused?

Answer: Yes…but…

Ten Tenets of Testing

1. Test single complete scenarios
2. Do not create dependencies between tests
3. Only verify a single thing in each assertion
4. Respect class encapsulation
5. Test limit values and boundaries
6. Test expected exceptional scenarios
7. Test interactions with other objects
8. When you find a bug, write a test to show it
9. Do not duplicate application logic in tests
10. Keep your test code clean

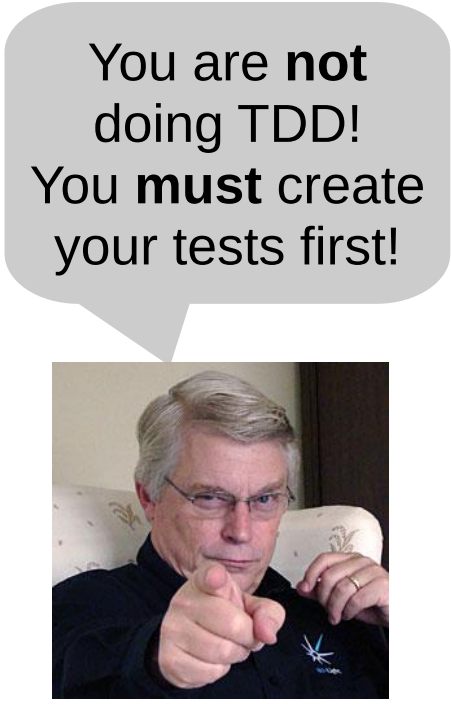
Ten Tenets of Testing +

Tenent 0

Verify tests are correct!

- Change code to break tests.
- Read the code and tests.
- Create a variant of test.
- Remove part of a test.

Question: Do I always have to write tests first?



Common Belief

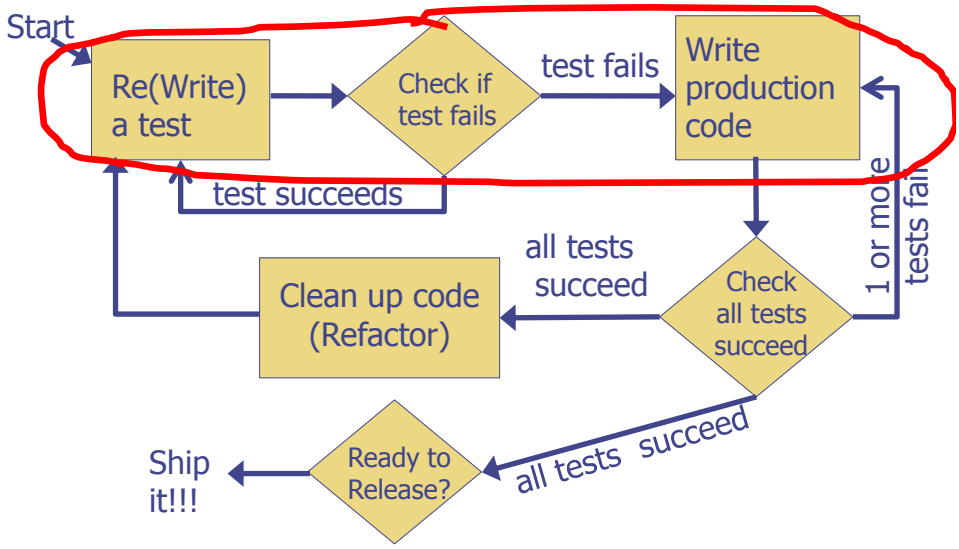
- You are **not** practicing **TDD correctly** unless you **write tests first**, before writing any code that is tested.

Is it OK to write tests after you write production code?

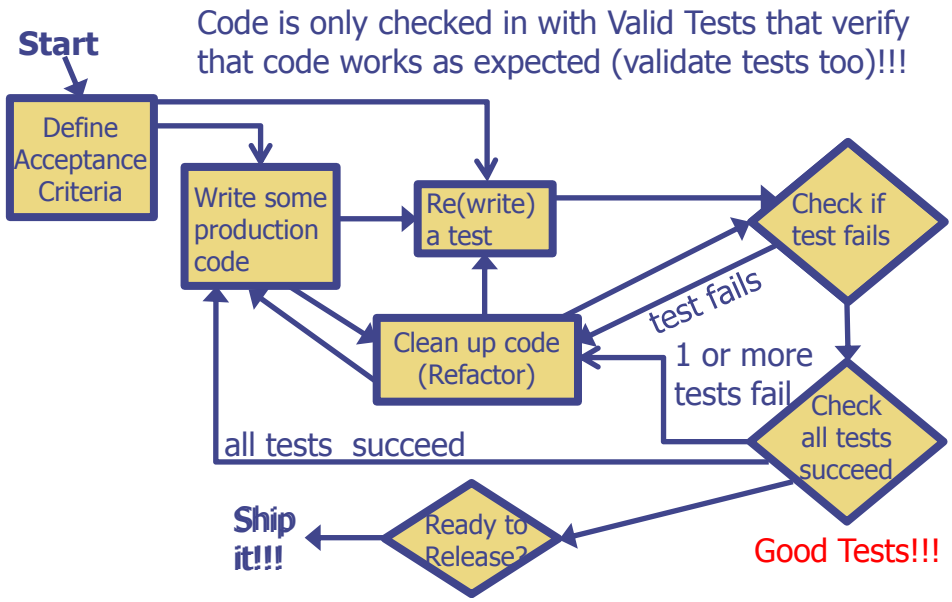
- Is this cheating?
- Does this result in bad code?
- Does it result in a bad design?
- Does it reinforce "slacker" tendencies to not write tests?

Common Practice

Tests don't always get written first.

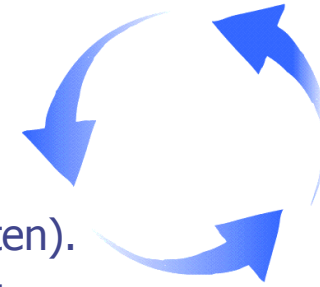


A Pragmatic Testing Cycle



Pragmatic Testing Cycle

Alternate between writing test code and production in small steps.



Use feedback from tests to verify code (integrate often).

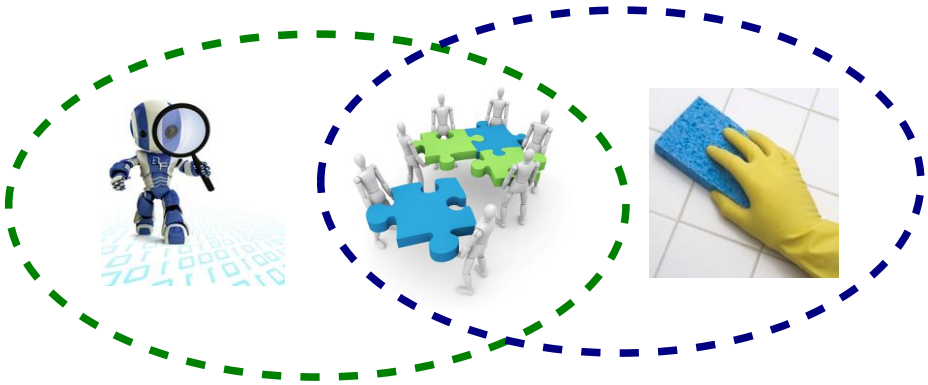
Don't worry whether the chicken or the egg comes first. Sometimes development guides testing.

Do what **yields** the most **value**!

Question: Do I always have to write tests first?

Answer: No, as long as tests are checked in with production code!

Question: Do tests allow you to change code safely and quickly?



Test-Driven Development

Refactoring

Common Wisdom

Work refactoring into your daily routine…

"In almost all cases, I'm opposed to setting aside time for refactoring. In my view refactoring is not an activity you set aside time to do. **Refactoring** is something you **do all the time** in little bursts." — Martin Fowler

Two Refactoring Types*

- Floss Refactorings—frequent, small changes, intermingled with other programming (daily health)

- Root canal refactorings — infrequent, protracted refactoring, during which programmers do nothing else (major repair)

* Emerson Murphy-Hill and Andrew Black in "Refactoring Tools: Fitness for Purpose"
http://web.cecs.pdx.edu/~black/publications/IEEESoftwareRefact.pdf

What if the Tests are an Obstacle to Refactoring?

The design of test code needs to evolve, just like production code!

Sometimes it is easier to throw away tests, change the design of your production code, and then write new tests.

Question: Do tests allow you to change code safely and quickly?

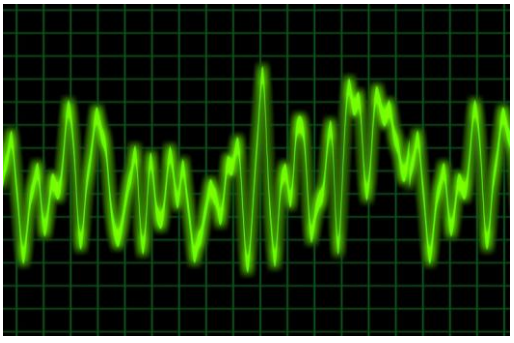
Answer: Yes and no.

Question: Do tests help with quality code and good design?

Agile Design Values

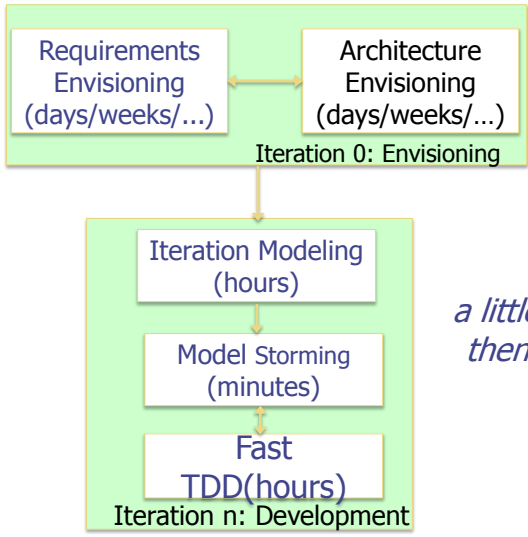
- Core values:
  - Design Simplicity
  - Communication
  - Teamwork
  - Trust
  - Satisfying stakeholder needs
- Keep learning
- **Lots of Testing!!!**

Classic Test-Driven Development Rhythm



- User story-by-story:
  - Write the simplest test
  - Run the test and fail
  - Write the simplest code that will pass the test
  - Run the test and pass
- Repeat until a "story" is tested and implemented
- *"Design happens between the keystrokes"*

Another View of Test-Driven Development



Conceptual Modeling

a little bit of modeling then a lot of coding

Spike Solutions

If some technical difficulty threatens to hold up the system's development,

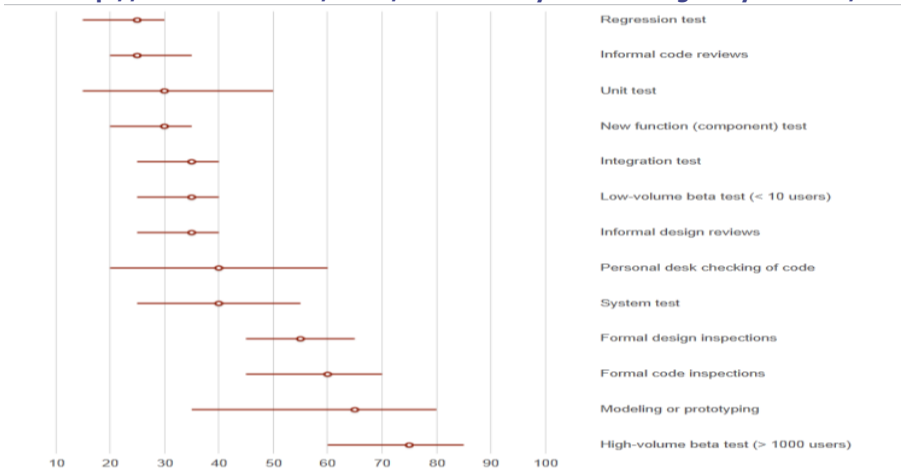
Or you are not sure how to solve a particular problem…

"Put a pair of developers on the problem for a week or two to reduce the potential risk"

Other Techniques for Improving Quality

Steve McConnell

http://kev.inburke.com/kevin/the-best-ways-to-find-bugs-in-your-code/



Combine and Conquer

- The average is 40% for any one technique…
- No one approach is adequate
- Combining techniques gives you much higher quality (> 90%)

Question: Do tests help with quality code and good design?

Answer: They can but …
you need more than tests.

Some Agile Myths

- Simple solutions are always best.
- We can easily adapt to changing requirements (new requirements).
- Because I'm following TDD, I can reliably change the system fast!!!
- TDD will ensure good Design.

"www.agilemyths.com"

TDD Challenges

1. Unit Tests aren't enough… should they be the main focus?
2. It is hard to know the "right" tests.
3. Deception that incrementally writing tests inductively proves software works… probably more deductive and example driven…
4. Tests can constrain code evolution and refactoring.
5. Get the "Whole" team involved!!!

Dogmatic

Synonyms: **bullheaded, dictative**, **doctrinaire**, **fanatical**, **intolerant**

Antonyms: amenable, flexible, manageable

Pragmatic

Synonyms: **common**, **commonsense**, **logical**, **practical**, **rational**, **realistic**, **sensible**

Antonyms: idealistic, unrealistic

Being Pragmatic… TDD

- Having the right tests is most important
- Tests and code evolve together
- Tests enable design change
  - But should not slow it down
- Mix up both fast and slow thinking.
- Doesn't matter if test comes first
  - As long as you write tests
- Keep the Agile Mindset and use the Whole Team!

Resources

- Agile Myths: agilemyths.com
- The Refactory: www.refactory.com
- Joe's website: joeyoder.com
- Wirfs-Brock Associates: www.wirfs-brock.com
- Our Pragmatic TDD Course:
  - refactory.com/training/test-driven-development
  - wirfs-brock.com/pragmatictestdrivendevelopment.html
- Introducing Pragmatic TDD:
  - wirfs-brock.com/blog/2011/09/23/what-is-pragmatic-testing-all-about/
  - http://adaptiveobjectmodel.com/2012/01/what-is-pragmatic-tdd/

Where Did All The Time Go?

Dzięki!!!

rebecca@wirfs-brock.com
Twitter: @rebeccawb

joe@refactory.com
Twitter: @metayoda