

TDD is all about discipline

- 1. Write a test that describes the behavior your system should have.
- 2. Run all tests. The newly added test should fail (enter red phase).
- 3. Write the minimum amount of code that makes the test pass.
- 4. Run all test. The newly added test should pass (enter green phase).
- 5. Refactor to remove duplication.



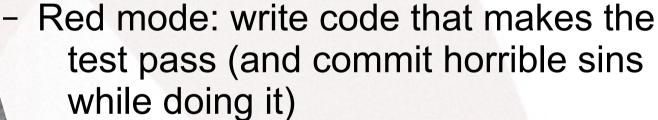
Why is TDD interesting?

- Breaks down the problem into small chunks.
 - Granularity is adjustable but teeny-tiny steps are encouraged.
- Focuses on requirements (verified by well-defined tests) before focusing on implementation
 - Based on the YAGNI (You Ain't Gonna Need It) principle
- Produces self-testing code that enables confident refactoring
- Goal: "Clean code that works"



Swapping between your red and green hats

- Focus on implementing functionality or producing clean design, each at a time
- This implies transitioning rhythmically between these 2 modes:



Green mode: remove duplication to create a code base that you can confidently take into the future



TDD is the zen of programming

- You just pay attention to the current state of the system
 - Emerging design
- By keeping to your code of discipline (the 5 steps), you prevent the mind from getting stuck or disconnected from the problem at hand
 - Management of fear and anxiety



Emerging design

- By focusing on the tests that verify the desired behavior, we consciously avoid tackling the design upfront
 - The consequence is an "evolving system" that reveals itself before our very eyes
- In reality, we decompose the "grand-design" into many small design decisions, based on the requirement at hand
 - Avoid solving problems which are not related to the behavior desired but to the design



Control your mind

- How many times have you been afraid of making a change for fear of breaking something?
- How many times have you thought of a solution that you "wanted" to be right and pushed it beyond a reasonable limit?
- How many times have you thought that you don't have time to test a change (or it's just too boring or not "your job")?
- How many times did you start a project excited and then you couldn't wait to get rid of it?



There is no silver bullet (1/2)

- TDD could seem like a method for dummies
- It's not: your wits are thoroughly tested in step 5 (refactor to remove duplication)
 - You have to know the programming paradigm
 - You have to know the language idioms
 - You have to know about tradeoff decisions
 - You have to know a fairly large refactoring catalogue and when they are prescribed
- The promise of "clean code that works" will only be as true as your ability in these skills



There is no silver bullet (2/2)

- TDD is a method to "build the system right"
- We can still fail to "build the right system"
- Extensions of the technique include
 - Acceptance Test-Driven Development (ATDD)
 - Behavior-Driven Development (BDD)
- Amenable to agile methodologies
 - User stories to drive the intended behavior



Don't forget to test!

- TDD is a technique for developers
- Unit testing is not all there is to testing
- Don't kill your testers just yet!
- Integration and system test should still be part of the development cycle
- Benefits of TDD are a front-loading of defects
 - Cheaper software by shortening release cycles



Let's get hacking!

- Mars Rover Kata
- Inspired by the experiment at https://blog.deltaforce.io/wordpress/running-a-coding-dojo-to-improve-python-tdd-skills/
- Clone the git repo at

https://github.com/adriantineo/MarsRoverKataTDD.git