Test-driven Development

Master Class

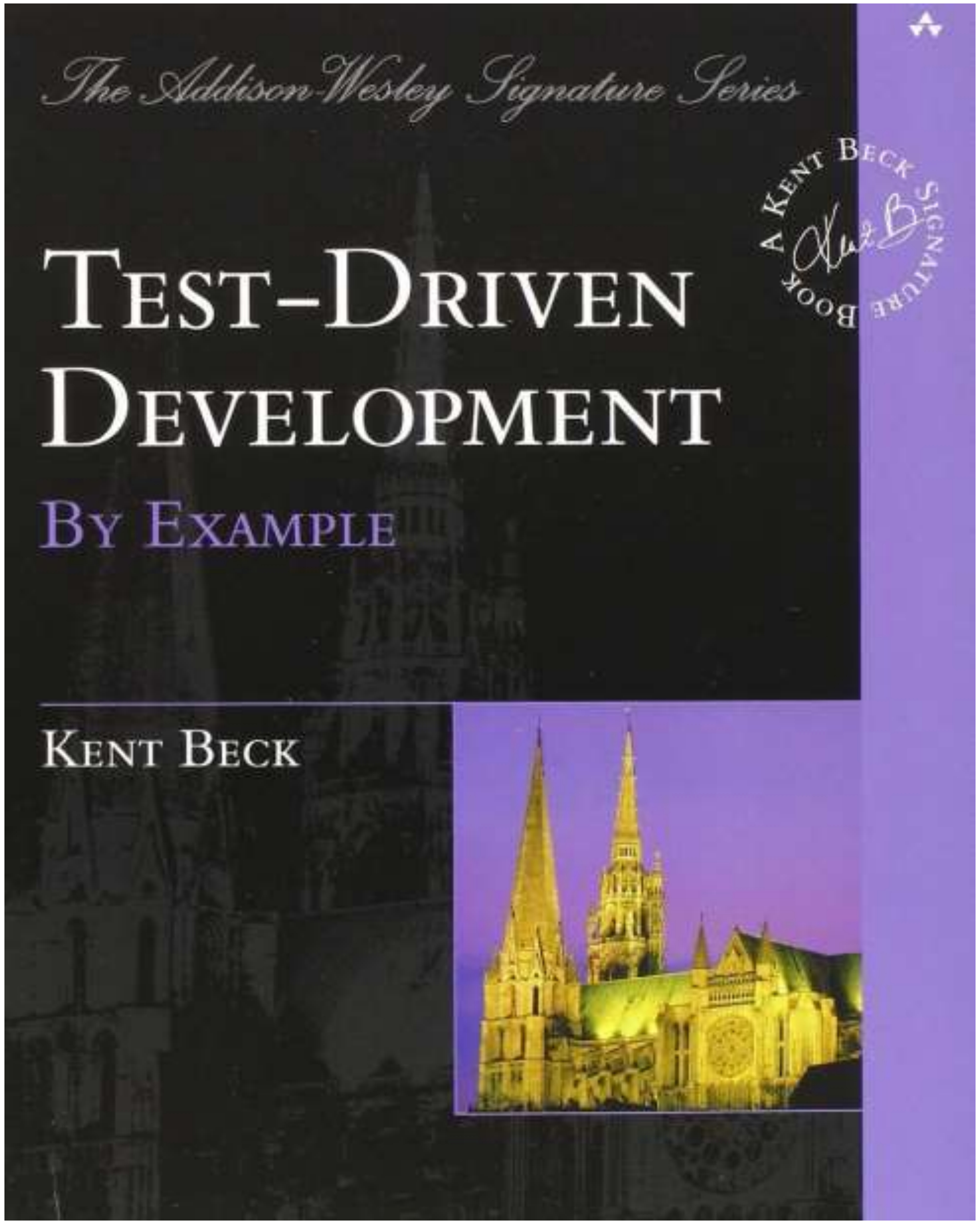
Day #1

What To Expect Today

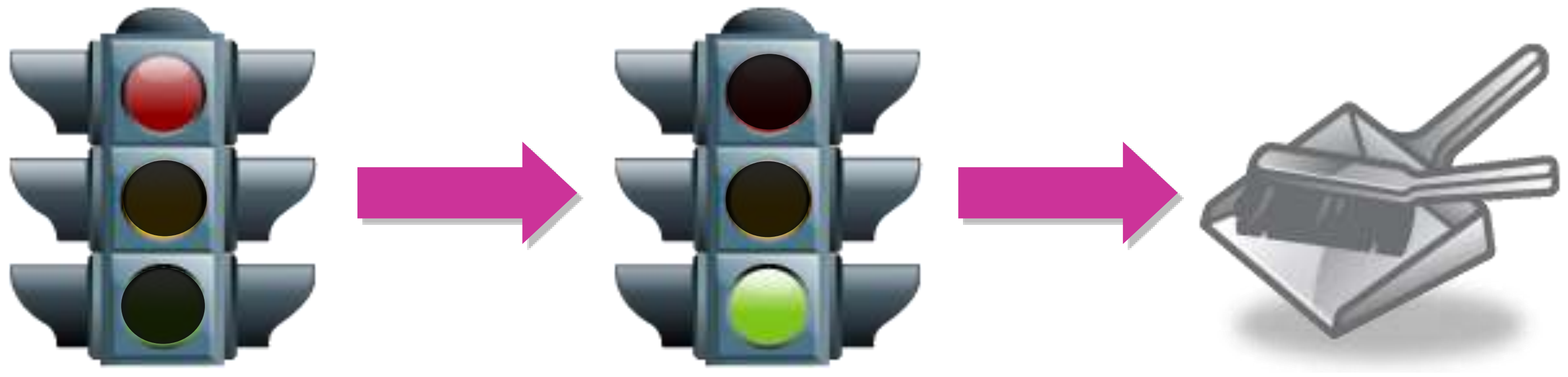
- Introduction to TDD
- Clean Code & Continuous Delivery
- TDD Basics
  - Write a failing test, write the simplest code to pass the test, refactor to remove duplication
- Example 1- Bank Transfer
- More TDD Basics
  - Write the assertion first and work backwards, see the test fail, write meaningful tests, triangulate
- Example 2 – Fibonacci Sequence Generator
- Another 3 TDD Basics
  - Keep your test and model code separate, isolate your tests, organise tests to reflect model code
- Example 3 – FizzBuzz
- Final 4 TDD Basics
  - Maintain your tests, tests should test one thing, don’t refactor when a test is failing
- Bonus Example 4 – FizzBuzzWhiz

What To Expect Tomorrow

- Test Doubles & Dependency Injection
- Example 5 – JG Holidays Ltd
- Putting It All Together:
  - Specification By Example & BDD
  - Test-driven Object Oriented Design
  - User Experience Design with Tests
- Example 6 – Community Video Library
- TDD On Legacy Code
- TDD Metrics
- TDD Practice Regimes


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck

Introduction To TDD

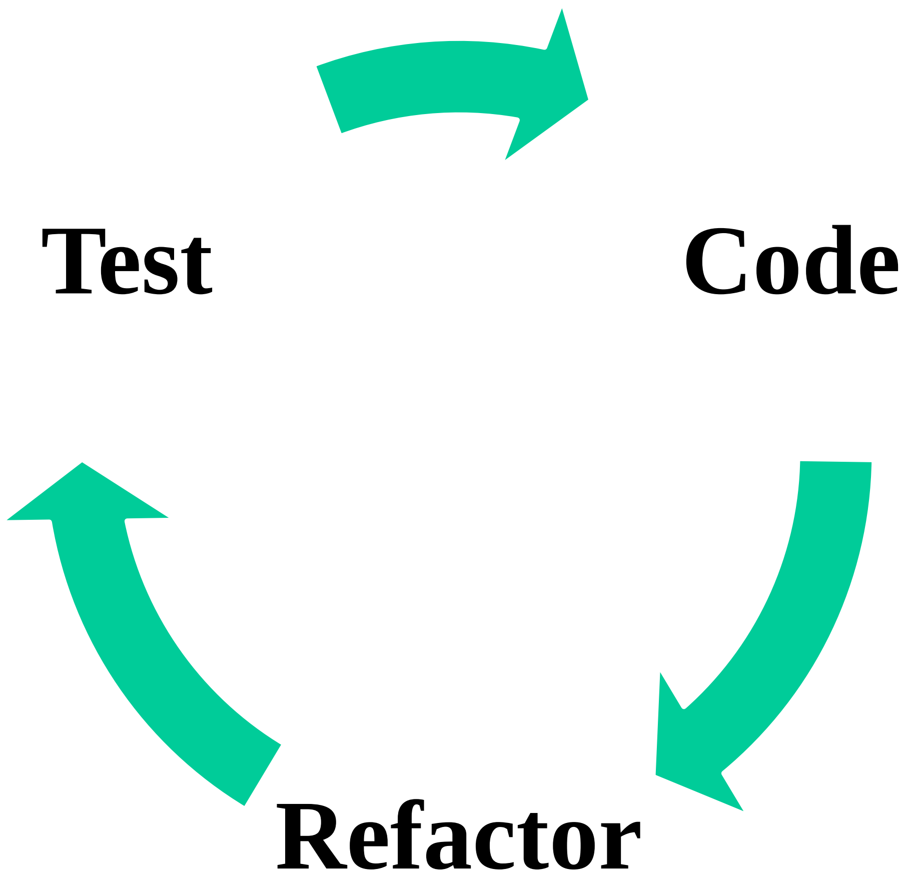


write a *failing* test

write the code to *pass* the test

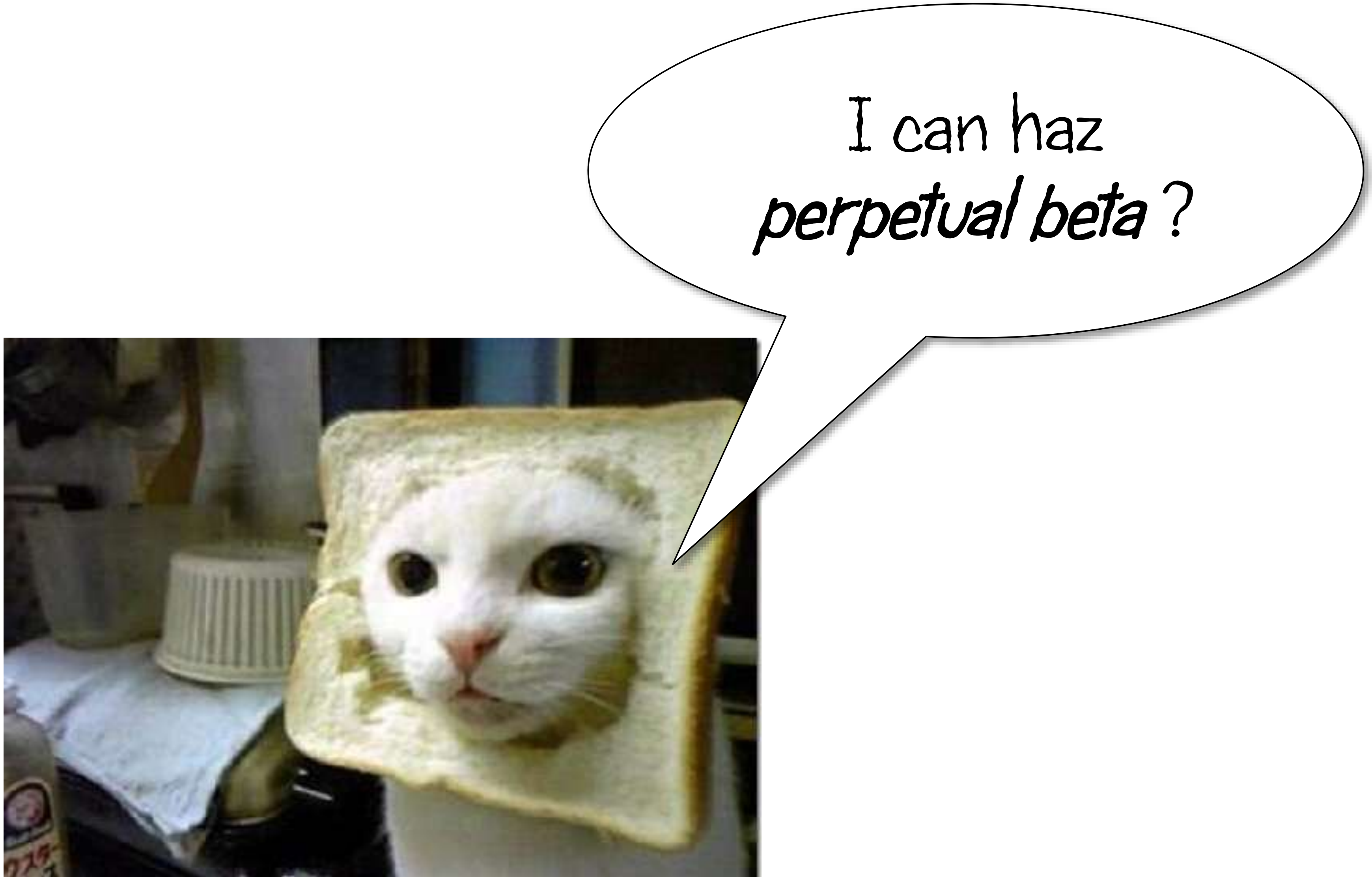
refactor to *remove duplication*

The TDD Cycle



CLEAN CODE & CONTINUOUS DELIVERY

codemanship



An Internet entrepreneur yesterday

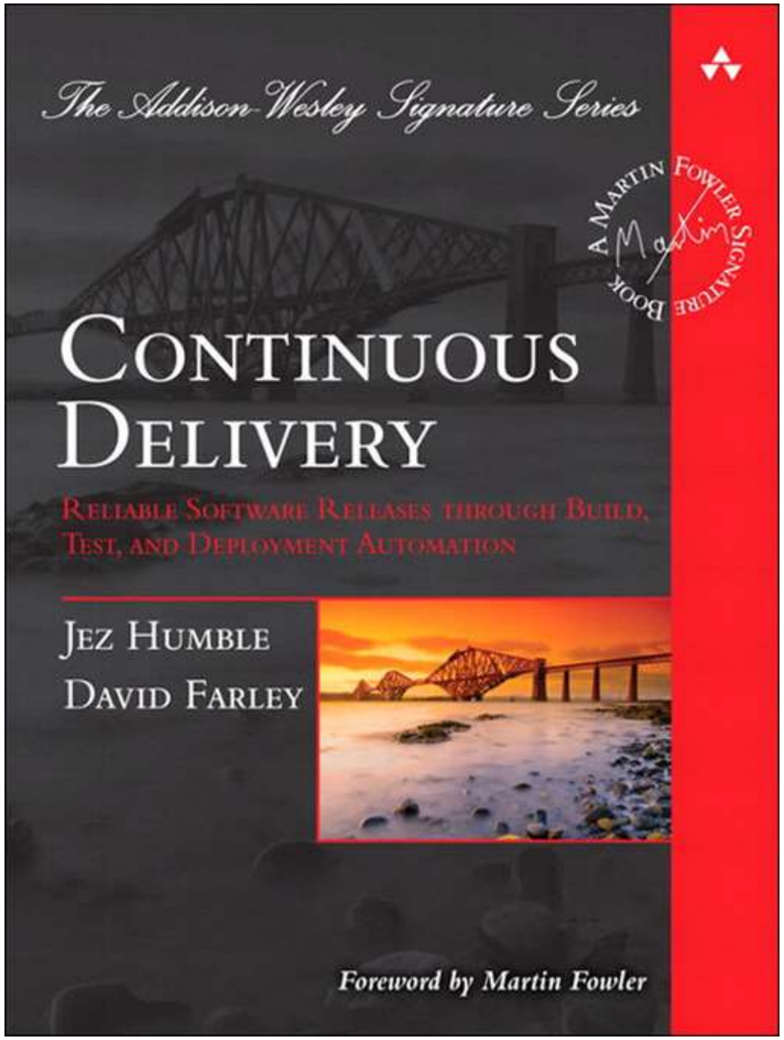
What It Is And How To Haz It

CONTINUOUS DELIVERY

codemanship

What Is Continuous Delivery?

Continuous delivery is the process of having a *shippable* product/ piece of software after each check in to source control.


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
Foreword by Martin Fowler

Business Implications Of Continuous Delivery

- Deployment becomes entirely a business decision
- Deployment can happen as frequently as the business requires
- Deployment can happen as soon as features & changes are ready
- At any given time, the amount of work in progress is minimised
- Cycle times from conception to delivery can be massively reduced
- Features/changes per release can be minimised
- Business feedback cycles can be minimised
- Businesses can **learn faster**

codemanship

Technical Implications of Continuous Delivery

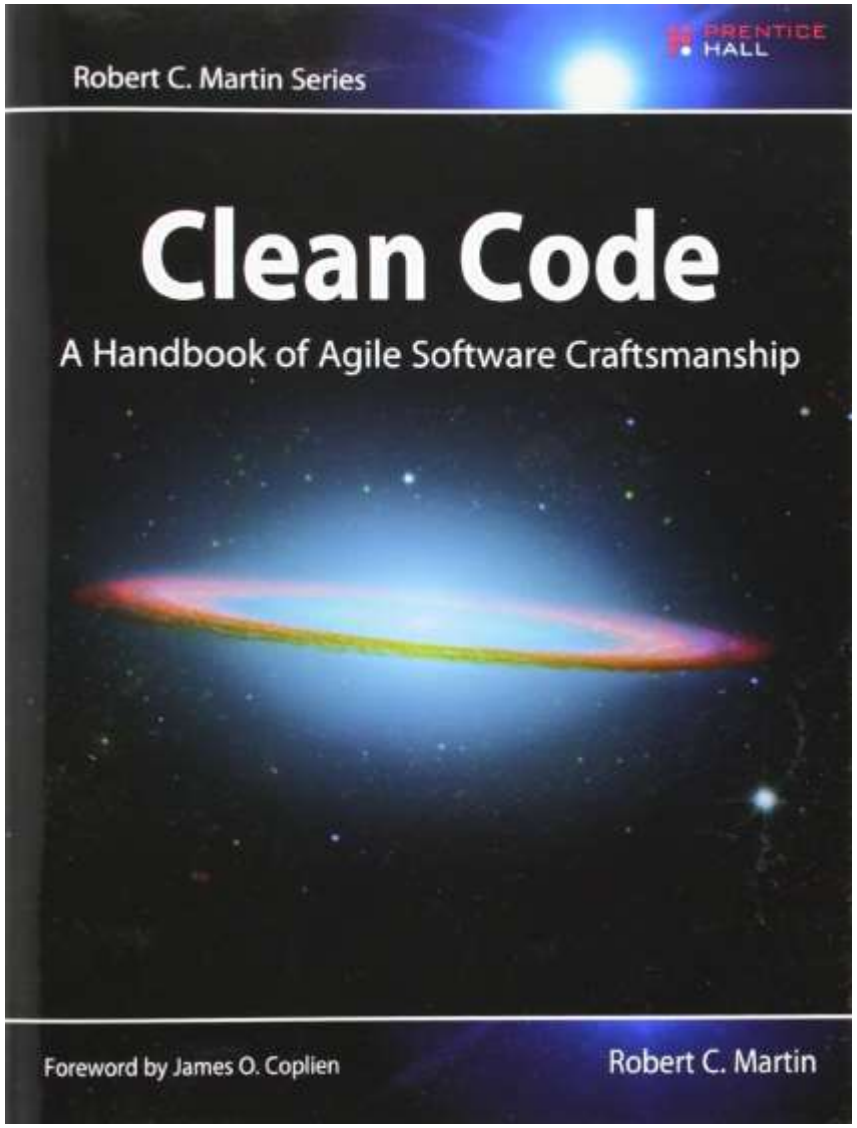
- The software must *always* be fit for purpose
- Test assurance needs to be **high**
- Programmers must be able to build and properly test the software quickly and cost-effectively
- Deploying software must be fast, reliable and cost-effective
- Building, testing & deploying software must be *fully automated*
- Whatever you deliver, it must be **easy to change** so it can continue to evolve

codemanship

I Can Haz Changes?

CLEAN CODE & CONTINUOUS DELIVERY

codemanship


Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

codemanship

Test Assurance

Readability

codemanship

Complexity

codemanship

Duplication

Hello, wrold!	Hello, wrold!	Hello, wrold!	Hello, wrold!
Hello, wrold!	Hello, wrold!	Hello, wrold!	Hello, wrold!
Hello, wrold!	Hello, wrold!	Hello, wrold!	Hello, wrold!

codemanship

Dependencies

codemanship

I Haz Skillz

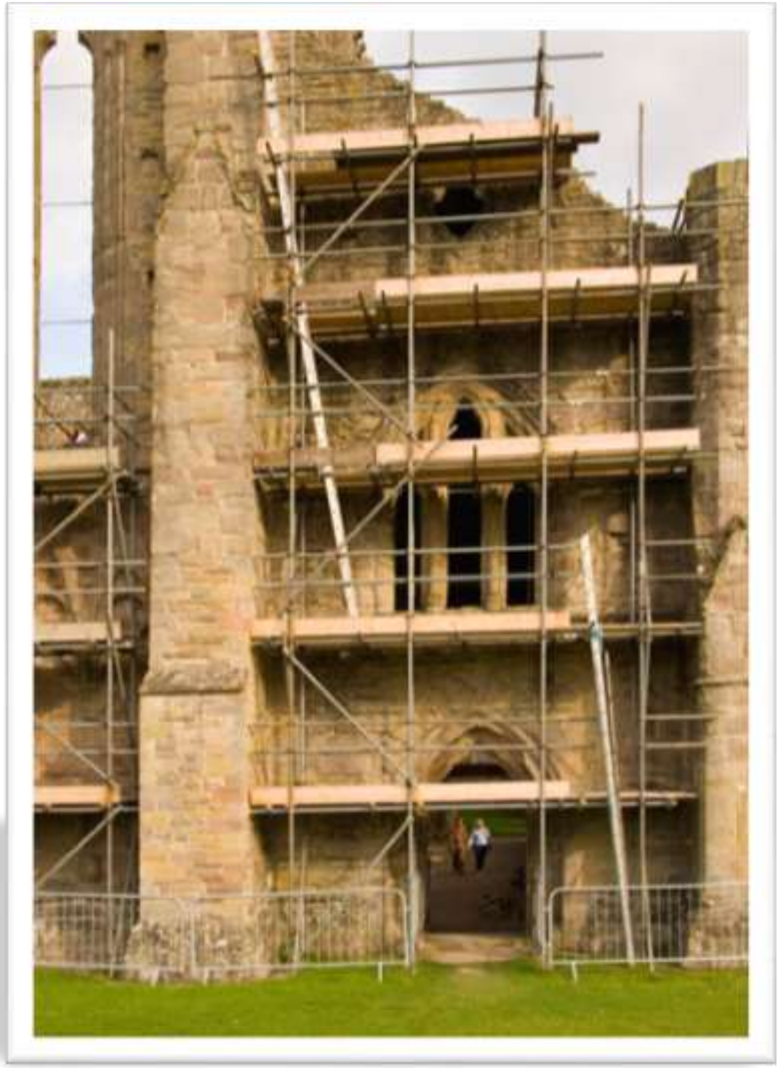
PROGRAMMER DISCIPLINES FOR CONTINUOUS DELIVERY

codemanship

Test-driven Development

codemanship

Refactoring



codemanship

Continuous Integration

codemanship

Continuous Inspection

codemanship

Continuous Improvement

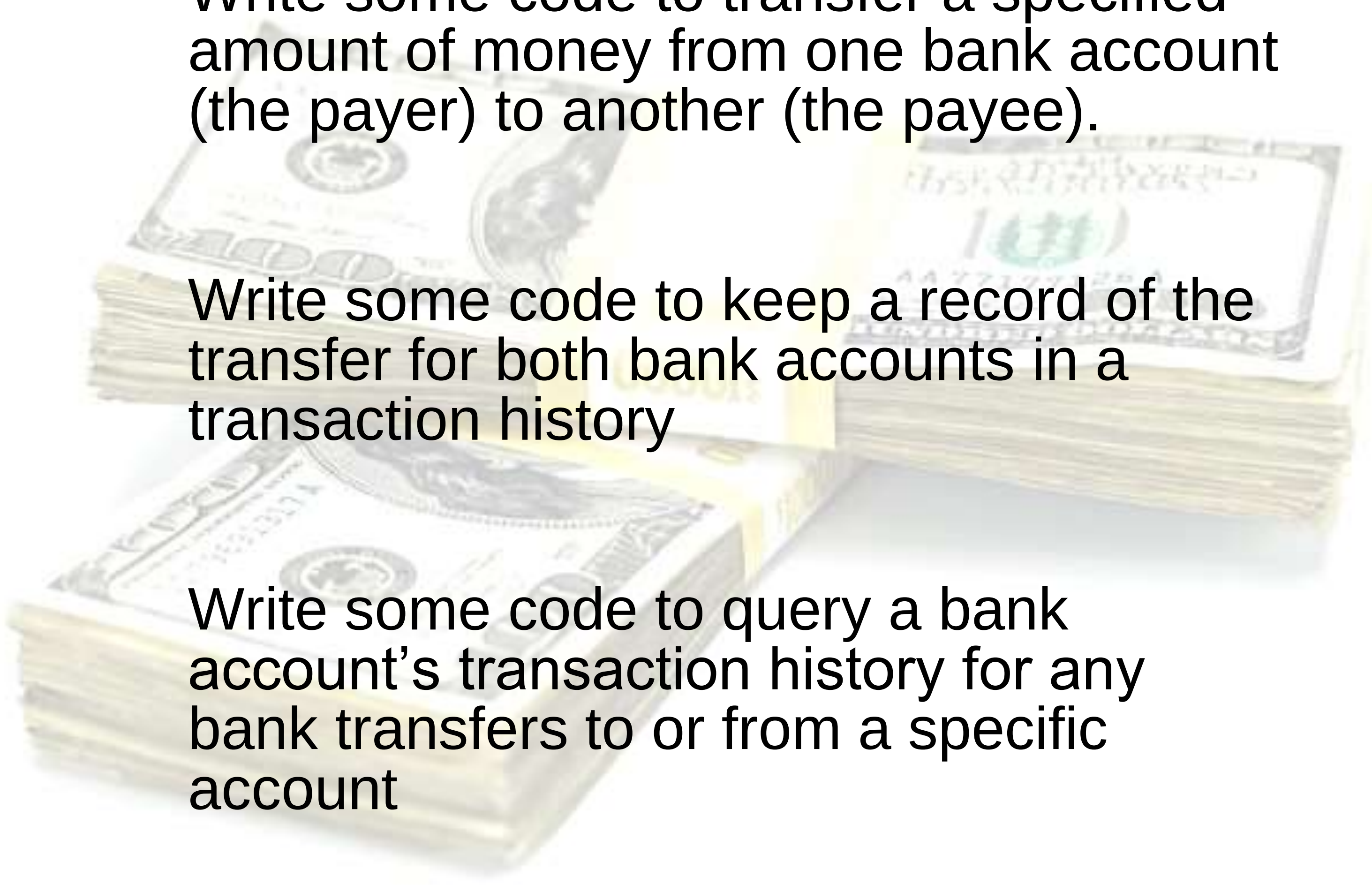
Automation

codemanship

Enough Talk…

LET’S GET CODING

Example #1 – Bank Transfer



Write some code to transfer a specified amount of money from one bank account (the payer) to another (the payee).

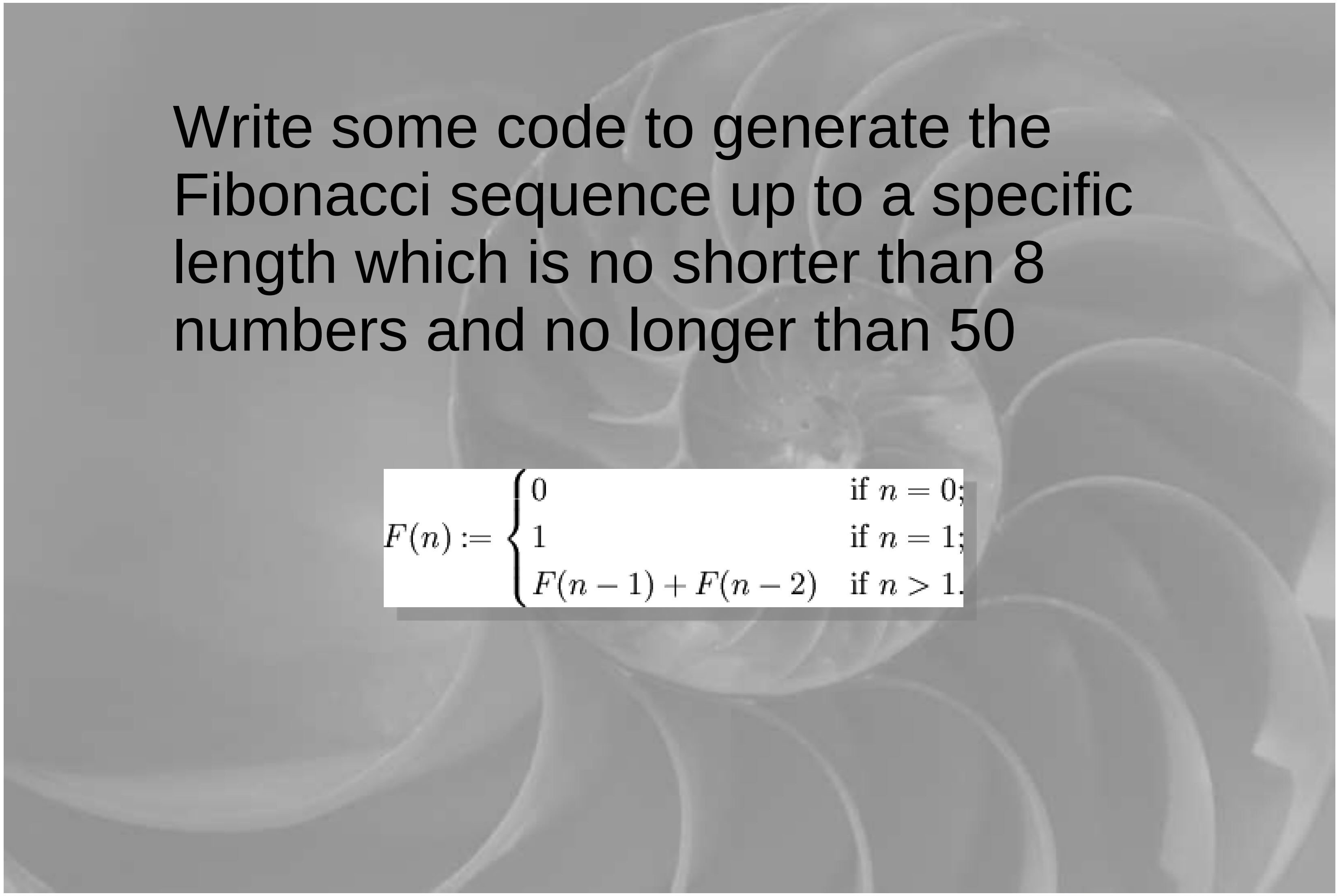
Write some code to keep a record of the transfer for both bank accounts in a transaction history

Write some code to query a bank account's transaction history for any bank transfers to or from a specific account

More TDD Basics

1. Write the assertion first and work backwards
2. Run the test to ensure it fails in the way you expect it to
3. Write meaningful tests that are self-explanatory
4. Triangulate through concrete examples towards general solutions

Example #2 – Fibonacci Sequence Generator



Write some code to generate the Fibonacci sequence up to a specific length which is no shorter than 8 numbers and no longer than 50

$$F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

Another 3 TDD Basics

1. Keep your test and model code separate
2. Isolate your tests so they run independently
3. Organise tests to reflect organisation of model code

Example #3 - FizzBuzz

Write some code that will generate a string of integers, starting at 1 and going up to 100, all separated by commas. Substitute any integer which is divisible by 3 with “Fizz”, and any integer which is divisible by 5 with “Buzz”, and any integer divisible by 3 and 5 with “FizzBuzz”

1,2,Fizz,4,Buzz,Fizz,7,8,Fizz,Buzz,11,Fizz,13,14,FizzBuzz… etc

codemanship

Final 4 TDD Basics

1. Maintain your tests
2. Tests should test one thing
3. Don’t refactor when a test is failing

Example #4 - FizzBuzzWhiz

Just as example #3, except substitute prime numbers with “Whiz”

1,Whiz,FizzWhiz,5,BuzzWhiz,Fizz,Whiz,8,Fizz,etc