ThoughtWorks STUDIOS mingle twist go

Agile Development Practices

A course from
ThoughtWorks Studios

Agenda

- Introductions
- Pair Programming
- Test-Driven Development
- Refactoring
- Mocking
- BDD and functional testing
- Continuous Integration
- Continuous Delivery
- Retrospective

ThoughtWorks STUDIOS

Who Are We?

Luca, Kelley and Vlad

ThoughtWorks STUDIOS

Who are you?

- Who are you?
- What do you do?
- Why are you here?

All in 30 seconds or less

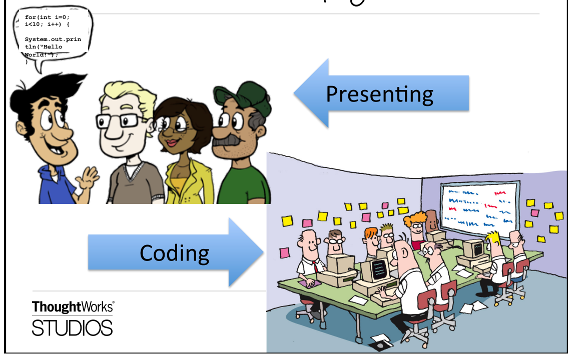
ThoughtWorks STUDIOS

Housekeeping

- Take phone calls outside team room
- Laptops down unless doing team work
- Questions as we go or in parking lot
- Break when needed
- Be on time
- Work in pairs

ThoughtWorks STUDIOS

Housekeeping



```
for(int i=0;
i<10; i++) {
System.out.prin
tln("Hello
World!");
```

ThoughtWorks STUDIOS

Presenting

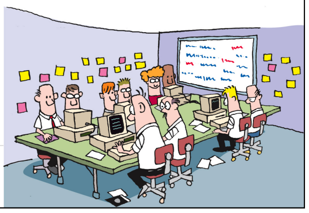
for(int i=0; i<10; i++) { System.out.println("Hello World"); }

- One conversation
- Attention to the speaker
- Laptops closed or put aside

ThoughtWorks STUDIOS

Coding

- Several conversations
  - Use your inside voice!
- Laptops open (of course)



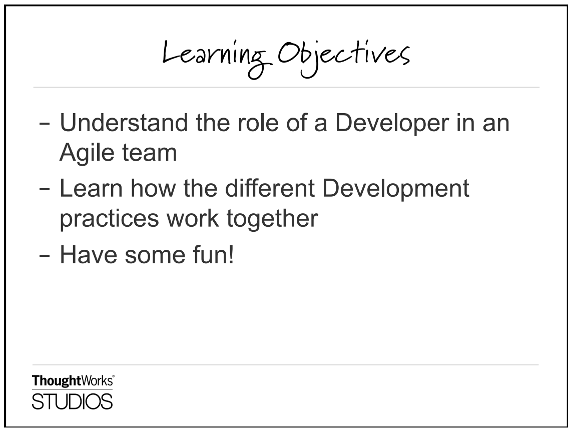
ThoughtWorks STUDIOS

Why Are We Here?

Hopes

Concerns

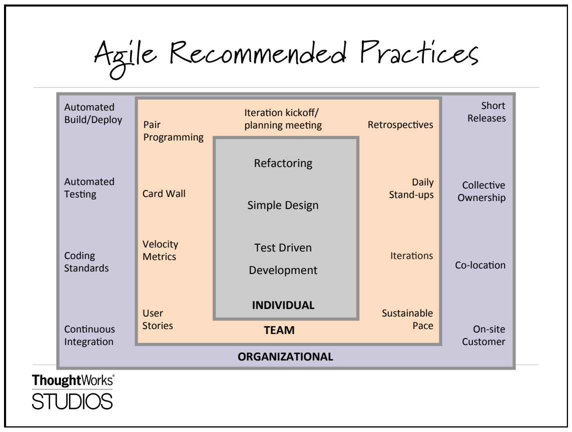
ThoughtWorks STUDIOS



Learning Objectives

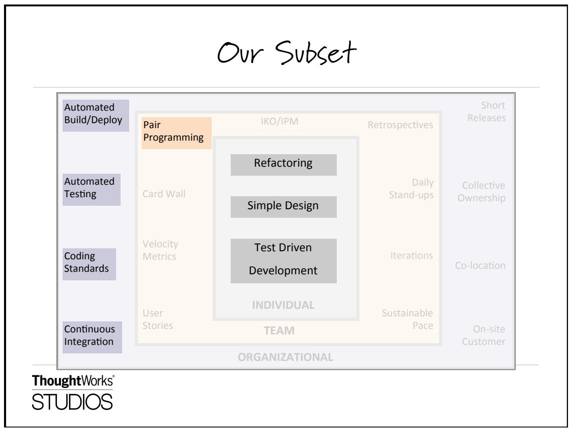
- Understand the role of a Developer in an Agile team
- Learn how the different Development practices work together
- Have some fun!

ThoughtWorks STUDIOS



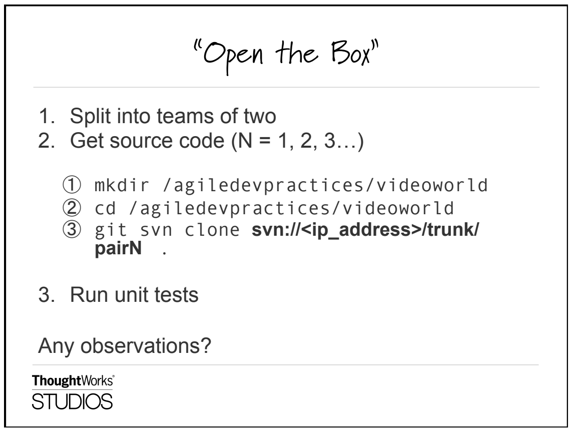
Agile Recommended Practices

ThoughtWorks STUDIOS



Our Subset

ThoughtWorks STUDIOS



"Open the Box"

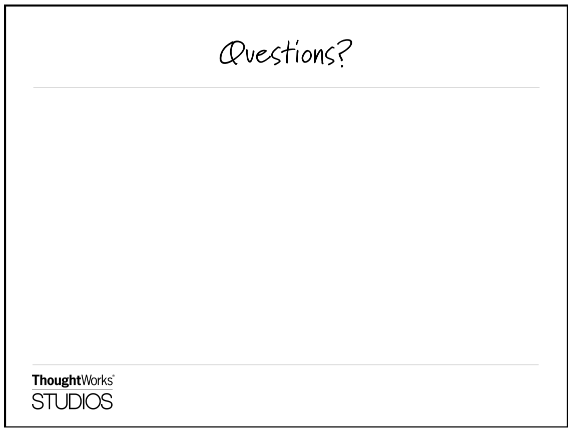
1. Split into teams of two
2. Get source code (N = 1, 2, 3…)

   ① `mkdir /agiledevpractices/videoworld`
   ② `cd /agiledevpractices/videoworld`
   ③ `git svn clone` **svn://<ip_address>/trunk/ pairN** .

3. Run unit tests

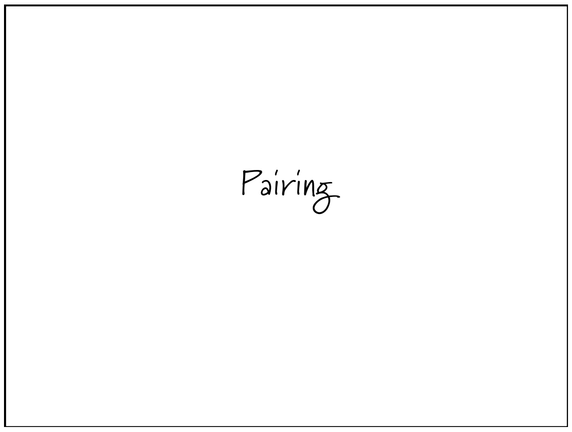
Any observations?

ThoughtWorks STUDIOS

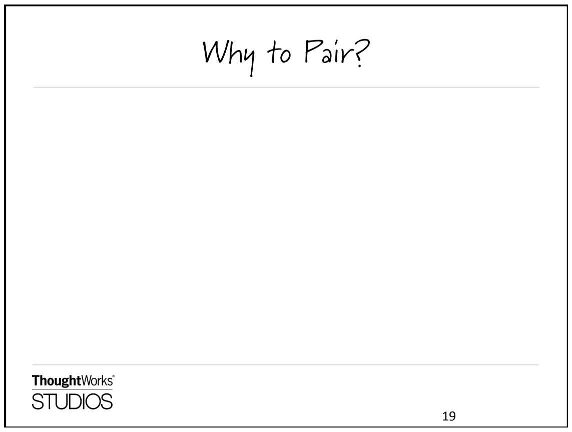


Questions?

ThoughtWorks STUDIOS



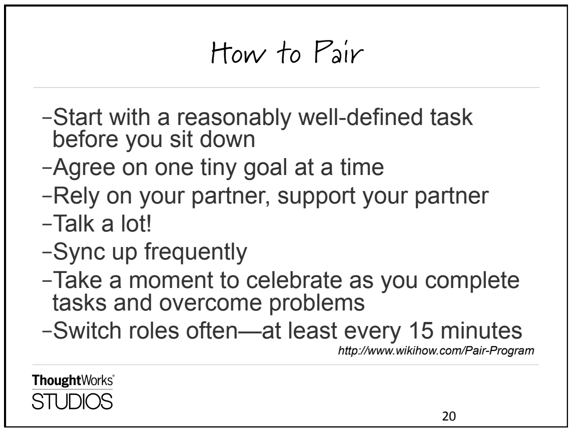
Pairing



Why to Pair?

ThoughtWorks STUDIOS

19



How to Pair

- Start with a reasonably well-defined task before you sit down
- Agree on one tiny goal at a time
- Rely on your partner, support your partner
- Talk a lot!
- Sync up frequently
- Take a moment to celebrate as you complete tasks and overcome problems
- Switch roles often—at least every 15 minutes

http://www.wikihow.com/Pair-Program

ThoughtWorks STUDIOS

20

Pairing Infrastructure

Monitors

One Monitor
- Both Pairs are looking at the same location
- Smaller Desktop

Two Monitors – Mirrored
- Pairs not looking at the same location
- Smaller Desktop
- Conversations don't come as easily

Two Monitors – Spanned
- Both Pairs are looking at the same location
- One Huge Desktop
- Enables more non-verbal communication

Keyboard and Mouse

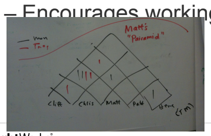
One Shared
- Encourages Communication
- Must either express yourself or ask for the keyboard

Two (One for each person)
- Faster to take control
- Less physical movement
- Less germ spreading!

Pair Rotation Techniques

Egg Timer / Chess Clock
- Encourages rotation of roles within a pair
- Driver swapping

Pair Stair/Pairamid
- Encourages working



H								
G								1
F							2	3
E						1	3	
D					2			
C				1				
B			2	3				
A		1	3					2
	A	B	C	D	E	F	G	H

Pairing Styles

Pairing Styles

Driver & Navigator

- Driver works tactically, Navigator works strategically
- Easiest to start with
- Roles should be swapped as frequently as possible

ThoughtWorks STUDIOS

26

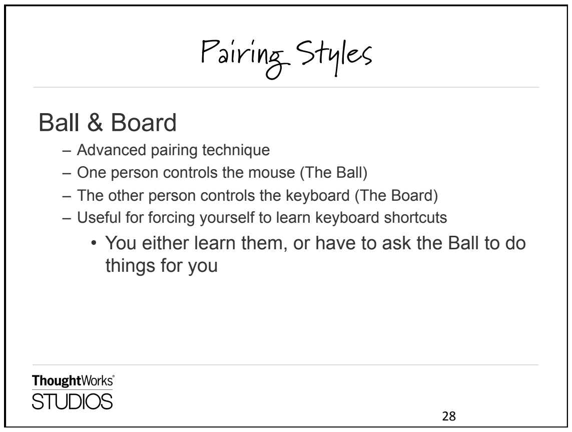
Pairing Styles

Ping-Pong

- 'A' writes a test, 'B' makes it green.
- Next 'B' writes the test, 'A' makes it green
- A mix of Pairing and Test Driven Development (TDD)

ThoughtWorks STUDIOS

27



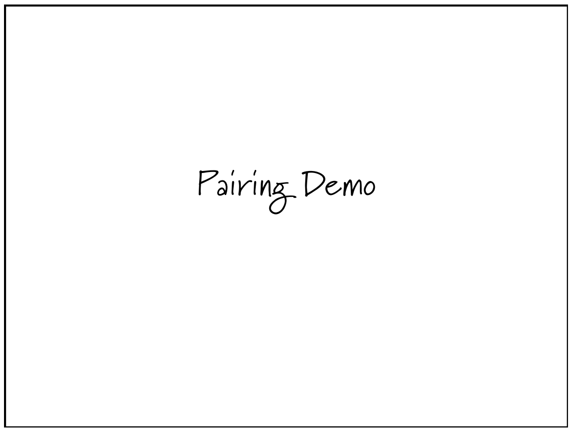
Pairing Styles

Ball & Board

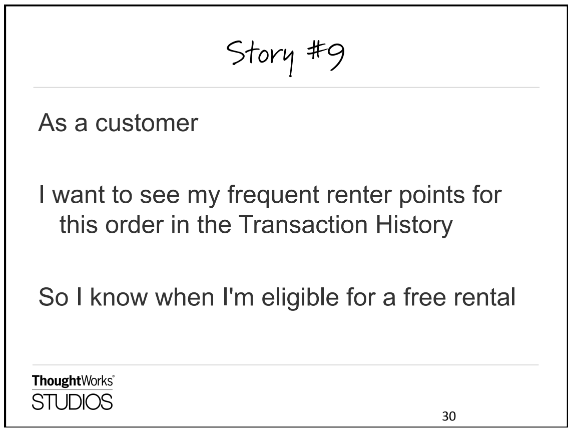
- Advanced pairing technique
- One person controls the mouse (The Ball)
- The other person controls the keyboard (The Board)
- Useful for forcing yourself to learn keyboard shortcuts
  - You either learn them, or have to ask the Ball to do things for you

ThoughtWorks STUDIOS

28



Pairing Demo



Story #9

As a customer

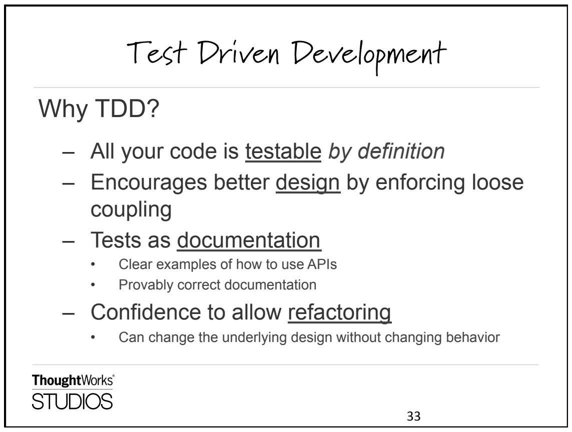
I want to see my frequent renter points for this order in the Transaction History

So I know when I'm eligible for a free rental

ThoughtWorks STUDIOS

30

Test Driven Development



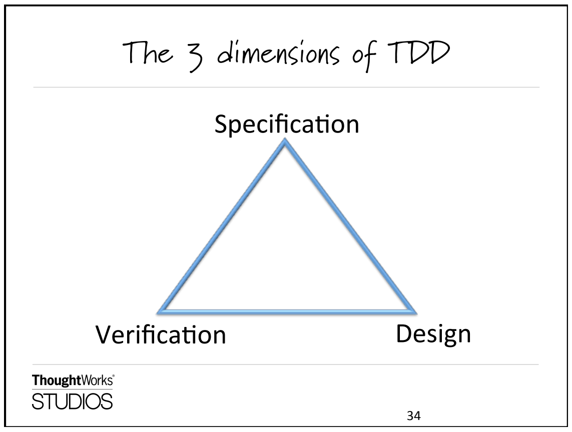
Test Driven Development

Why TDD?

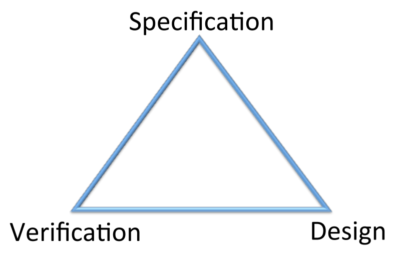
- All your code is testable *by definition*
- Encourages better design by enforcing loose coupling
- Tests as documentation
  - Clear examples of how to use APIs
  - Provably correct documentation
- Confidence to allow refactoring
  - Can change the underlying design without changing behavior

ThoughtWorks STUDIOS

33

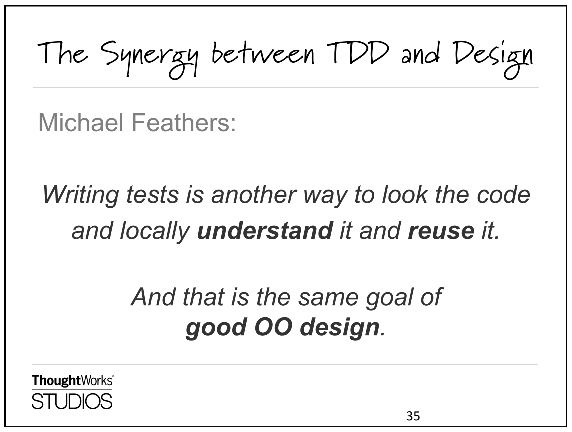


The 3 dimensions of TDD



ThoughtWorks STUDIOS

34



The Synergy between TDD and Design

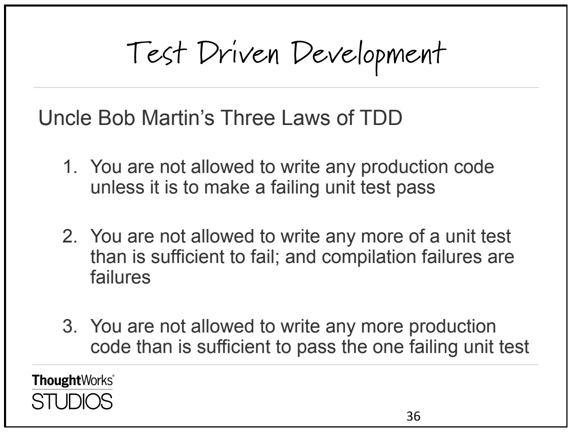
Michael Feathers:

*Writing tests is another way to look the code and locally **understand** it and **reuse** it.*

*And that is the same goal of **good OO design**.*

ThoughtWorks STUDIOS

35



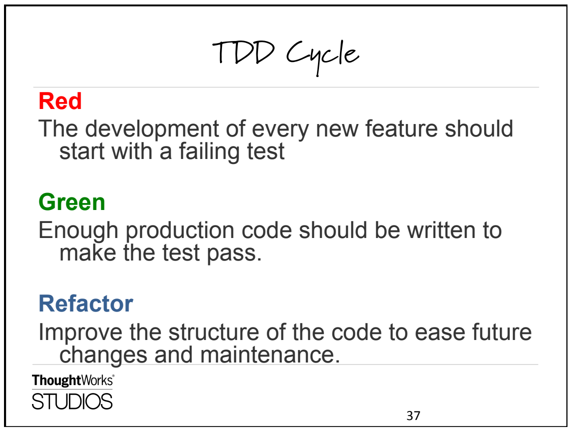
Test Driven Development

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

ThoughtWorks STUDIOS

36



TDD Cycle

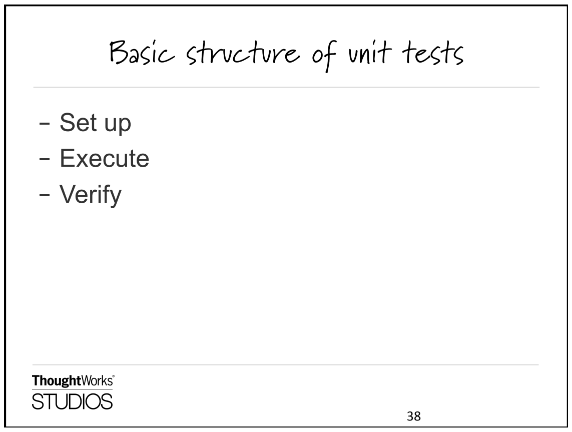
Red
The development of every new feature should start with a failing test

Green
Enough production code should be written to make the test pass.

Refactor
Improve the structure of the code to ease future changes and maintenance.

ThoughtWorks STUDIOS

37

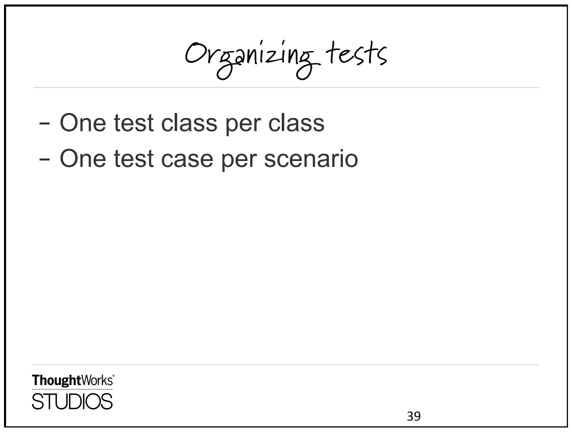


Basic structure of unit tests

- Set up
- Execute
- Verify

ThoughtWorks STUDIOS

38

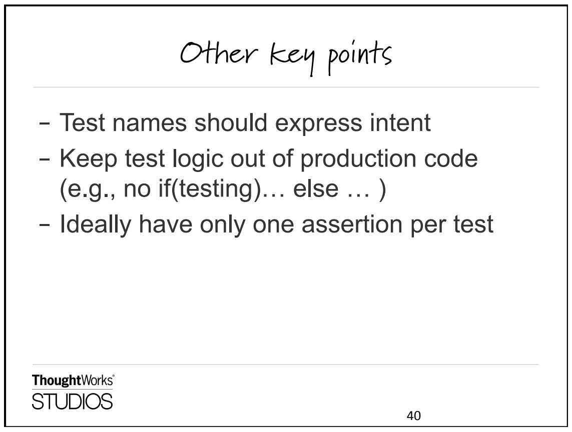


Organizing tests

- One test class per class
- One test case per scenario

ThoughtWorks STUDIOS

39

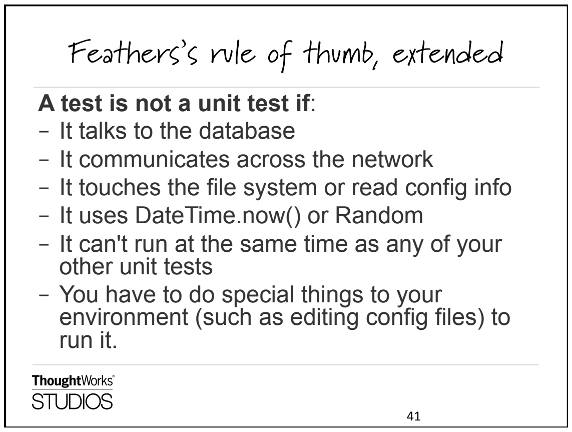


Other key points

- Test names should express intent
- Keep test logic out of production code (e.g., no if(testing)… else …)
- Ideally have only one assertion per test

ThoughtWorks STUDIOS

40



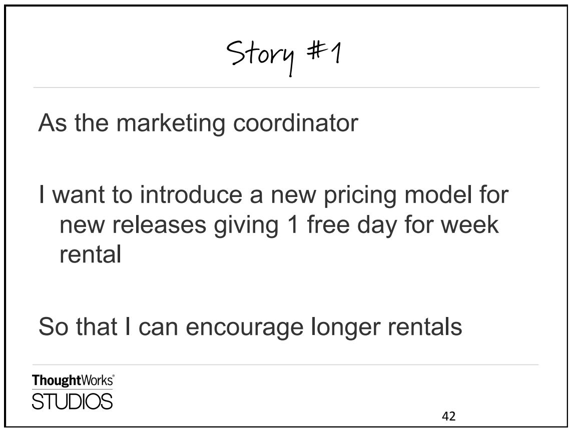
Feathers's rule of thumb, extended

A test is not a unit test if:

- It talks to the database
- It communicates across the network
- It touches the file system or read config info
- It uses DateTime.now() or Random
- It can't run at the same time as any of your other unit tests
- You have to do special things to your environment (such as editing config files) to run it.

ThoughtWorks STUDIOS

41



Story #1

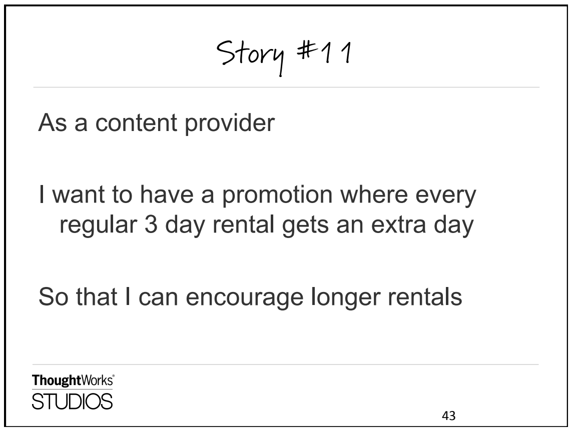
As the marketing coordinator

I want to introduce a new pricing model for new releases giving 1 free day for week rental

So that I can encourage longer rentals

ThoughtWorks STUDIOS

42



Story #11

As a content provider

I want to have a promotion where every regular 3 day rental gets an extra day

So that I can encourage longer rentals

ThoughtWorks STUDIOS

43

Refactoring

Refactoring

What it is
- The art of improving software readability and design of existing code without changing behavior
- A series of *small* steps
- Refactoring changes the balance point between up-front design and emergent design.

What it is not
- rewriting from scratch

ThoughtWorks STUDIOS

45

Some history

- William Opdyke - 1992
  - Refactoring Object-Oriented Frameworks[1], was the first in-depth study of code refactoring as a software engineering technique
  - Problem:- It is hard to change 'existing' OO programs.
  - Solution:- Can we automate 'code restructiring'?
- Martin Fowler
  - The famous 'Refactoring' book.
  - www.refactoring.com

ThoughtWorks STUDIOS

46

Smells

- Warning signs about potential problems in code.
- Sample smells
  - Unclear names of class, methods, variables, parameters
  - Comments
  - Long Method and large Class
  - Nested conditionals and loops
  - Code and Logic duplication

ThoughtWorks STUDIOS

47

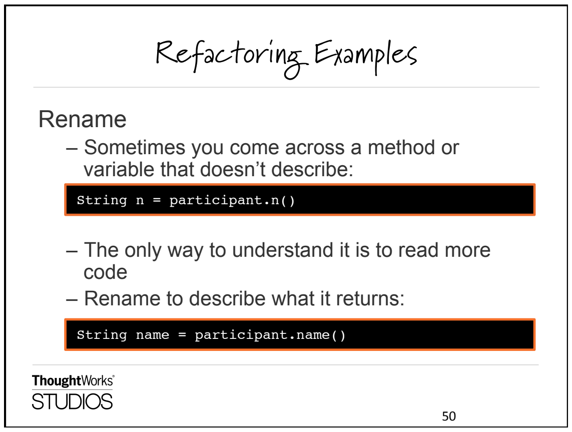
Refactoring Cycle

- Start with a working program.
- While smells remain:
  - Choose the worst smell.
  - Select a refactoring that will address the smell.
  - Apply the refactoring.

ThoughtWorks STUDIOS

48

Refactoring Examples



Refactoring Examples

Rename

– Sometimes you come across a method or variable that doesn't describe:

```
String n = participant.n()
```

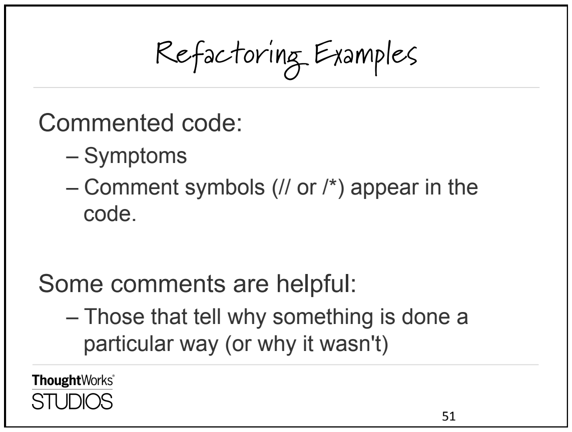
– The only way to understand it is to read more code

– Rename to describe what it returns:

```
String name = participant.name()
```

ThoughtWorks STUDIOS

50



Refactoring Examples

Commented code:

– Symptoms

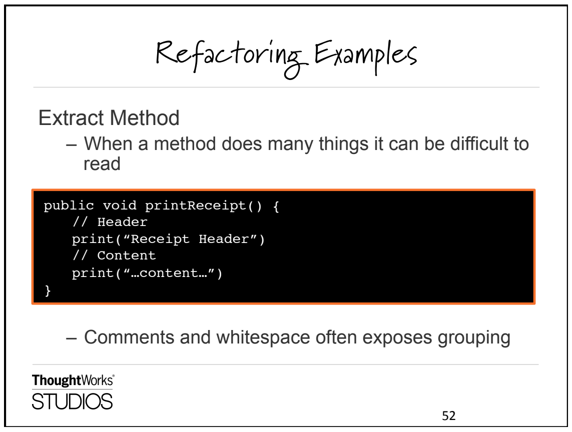
– Comment symbols (// or /*) appear in the code.

Some comments are helpful:

– Those that tell why something is done a particular way (or why it wasn't)

ThoughtWorks STUDIOS

51



Refactoring Examples

Extract Method

– When a method does many things it can be difficult to read

```
public void printReceipt() {
    // Header
    print("Receipt Header")
    // Content
    print("…content…")
}
```

– Comments and whitespace often exposes grouping

ThoughtWorks STUDIOS

52

Refactoring Examples

Extract Method

– When a method does many things it can be difficult to read

```
public void printReceipt() {
   printHeader()
   printContent()
}
private void printHeader() {
   print("Receipt Header")
}
private void printContent() {
   print("…content…")
}
```

ThoughtWorks STUDIOS

53

Split Loop

Split Loop

– Used to simplify logic that occurs in a loop

```
public void saveTotals(items) {
   for(item in items) {
      totalPoints += item.points
      totalCost += item.price
   }
}
```

ThoughtWorks STUDIOS

54

Refactoring Examples

Split Loop

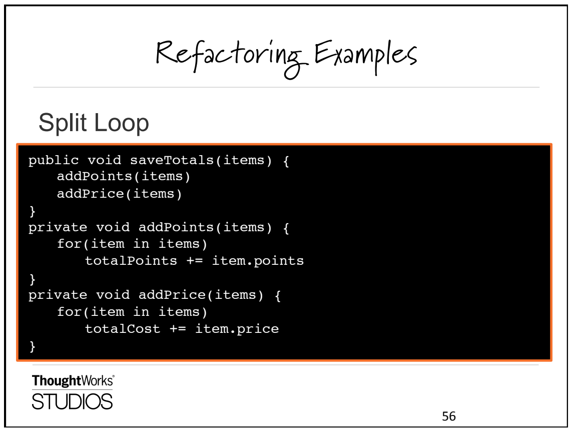
– Used to simplify logic that occurs in a loop

```
public void saveTotals(items) {
   for(item in items) {
      totalPoints += item.points
   }
   for(item in items) {
      totalCost += item.price
   }
}
```

– Now you can extract method to describe what they do

ThoughtWorks STUDIOS

55



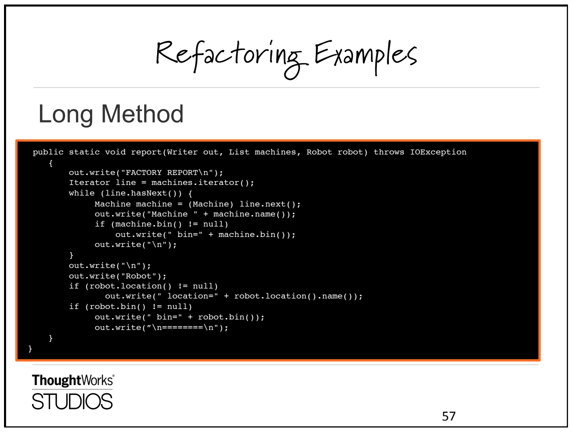
Refactoring Examples

Split Loop

```
public void saveTotals(items) {
    addPoints(items)
    addPrice(items)
}
private void addPoints(items) {
    for(item in items)
        totalPoints += item.points
}
private void addPrice(items) {
    for(item in items)
        totalCost += item.price
}
```

ThoughtWorks STUDIOS

56



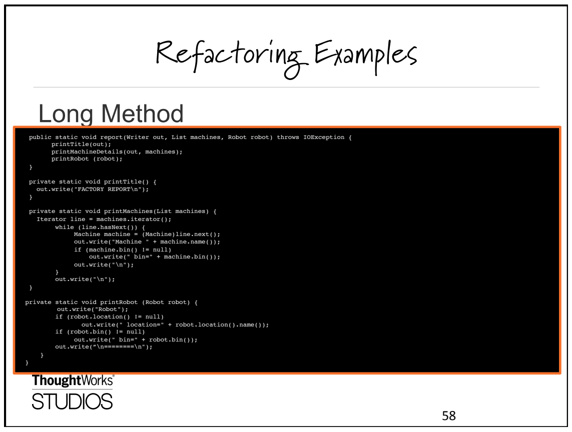
Refactoring Examples

Long Method

```
public static void report(Writer out, List machines, Robot robot) throws IOException
   {
        out.write("FACTORY REPORT\n");
        Iterator line = machines.iterator();
        while (line.hasNext()) {
             Machine machine = (Machine) line.next();
             out.write("Machine " + machine.name());
             if (machine.bin() != null)
                 out.write(" bin=" + machine.bin());
             out.write("\n");
        }
        out.write("\n");
        out.write("Robot");
        if (robot.location() != null)
               out.write(" location=" + robot.location().name());
        if (robot.bin() != null)
             out.write(" bin=" + robot.bin());
             out.write("\n========\n");
   }
}
```

ThoughtWorks STUDIOS

57



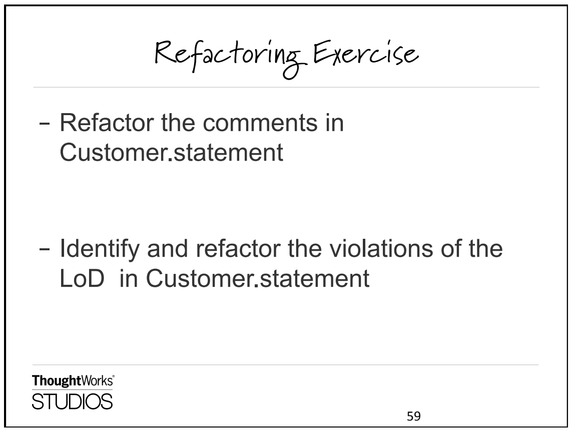
Refactoring Examples

Long Method

```
 public static void report(Writer out, List machines, Robot robot) throws IOException {
      printTitle(out);
      printMachineDetails(out, machines);
      printRobot (robot);
 }

 private static void printTitle() {
   out.write("FACTORY REPORT\n");
 }

 private static void printMachines(List machines) {
   Iterator line = machines.iterator();
        while (line.hasNext()) {
             Machine machine = (Machine)line.next();
             out.write("Machine " + machine.name());
             if (machine.bin() != null)
                 out.write(" bin=" + machine.bin());
             out.write("\n");
        }
        out.write("\n");
 }

private static void printRobot (Robot robot) {
        out.write("Robot");
        if (robot.location() != null)
               out.write(" location=" + robot.location().name());
        if (robot.bin() != null)
             out.write(" bin=" + robot.bin());
        out.write("\n========\n");
    }
}
```

ThoughtWorks STUDIOS

58

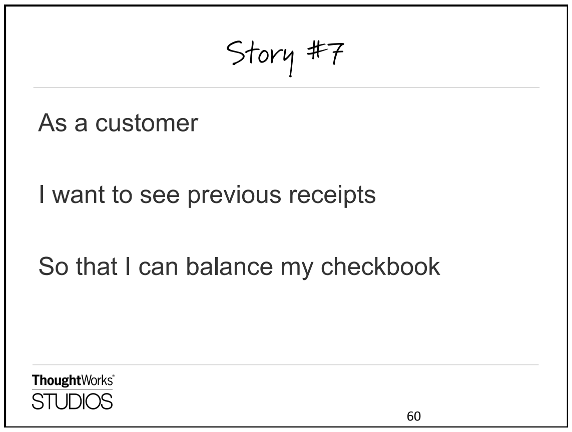


Refactoring Exercise

- Refactor the comments in Customer.statement
- Identify and refactor the violations of the LoD in Customer.statement

ThoughtWorks STUDIOS

59



Story #7

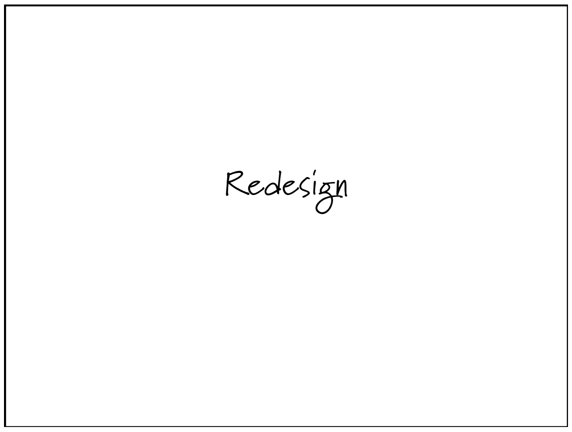
As a customer

I want to see previous receipts

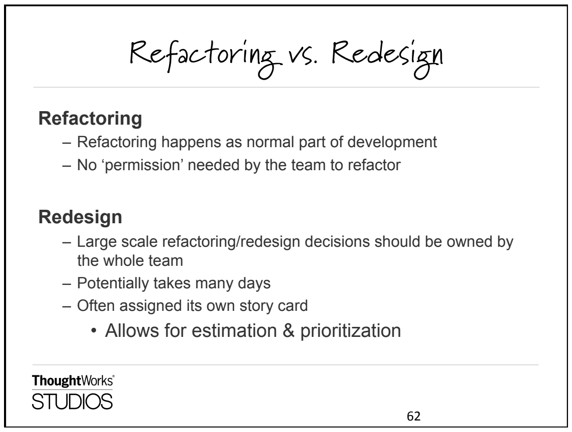
So that I can balance my checkbook

ThoughtWorks STUDIOS

60



Redesign

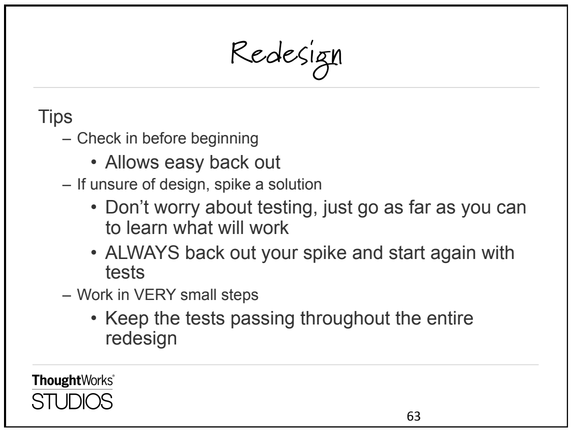


Refactoring vs. Redesign

Refactoring
- Refactoring happens as normal part of development
- No ‘permission’ needed by the team to refactor

Redesign
- Large scale refactoring/redesign decisions should be owned by the whole team
- Potentially takes many days
- Often assigned its own story card
  - Allows for estimation & prioritization

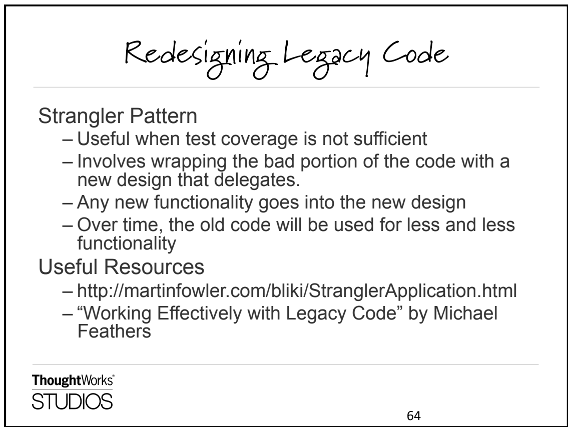
62



Redesign

Tips
- Check in before beginning
  - Allows easy back out
- If unsure of design, spike a solution
  - Don’t worry about testing, just go as far as you can to learn what will work
  - ALWAYS back out your spike and start again with tests
- Work in VERY small steps
  - Keep the tests passing throughout the entire redesign

63



Redesigning Legacy Code

Strangler Pattern
- Useful when test coverage is not sufficient
- Involves wrapping the bad portion of the code with a new design that delegates.
- Any new functionality goes into the new design
- Over time, the old code will be used for less and less functionality

Useful Resources
- http://martinfowler.com/bliki/StranglerApplication.html
- “Working Effectively with Legacy Code” by Michael Feathers

64

Story #10

As a customer

I want to see how much total I've spent

So I can manage my budget

ThoughtWorks STUDIOS

65

Mocking

Unit testing with test double

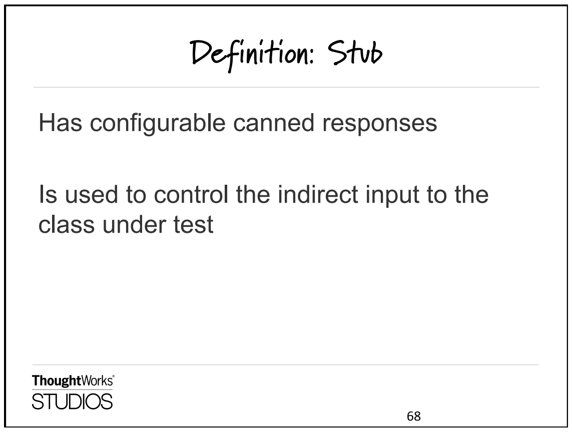
Definition: Test Double

In automated **unit testing** replaces an object on which the class under test depends

Can be a **Stub**, a **Mock** a **Spy**

ThoughtWorks STUDIOS

67



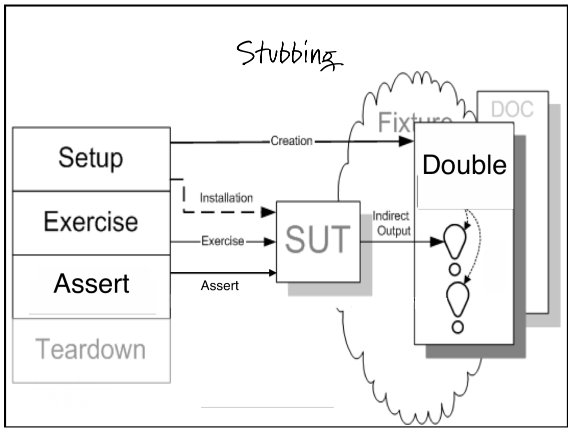
Definition: Stub

Has configurable canned responses

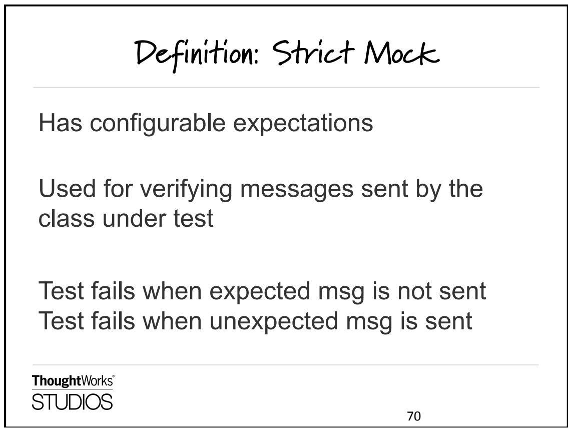
Is used to control the indirect input to the class under test

ThoughtWorks STUDIOS

68



Stubbing



Definition: Strict Mock

Has configurable expectations

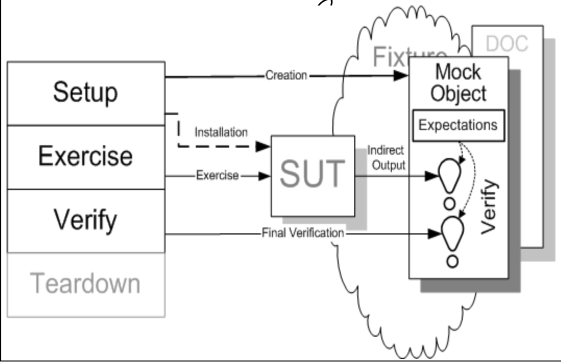
Used for verifying messages sent by the class under test

Test fails when expected msg is not sent
Test fails when unexpected msg is sent

ThoughtWorks STUDIOS

70

Mocking



When do you use Stubs?

- Code you are testing has external dependencies as
  - File System
  - Config info
  - Database
  - Network
    - E.g. Web services
  - Behavior you don't have control on
    - E.g. DateTime.now() or Random

ThoughtWorks STUDIOS

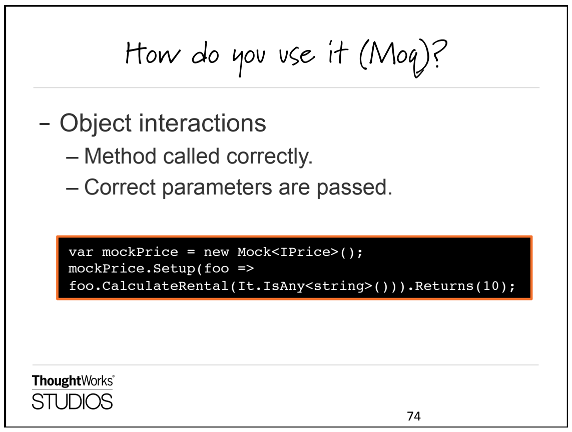
72

When do you use Mocks?

- Methods that return void and don't change visible state

ThoughtWorks STUDIOS

73



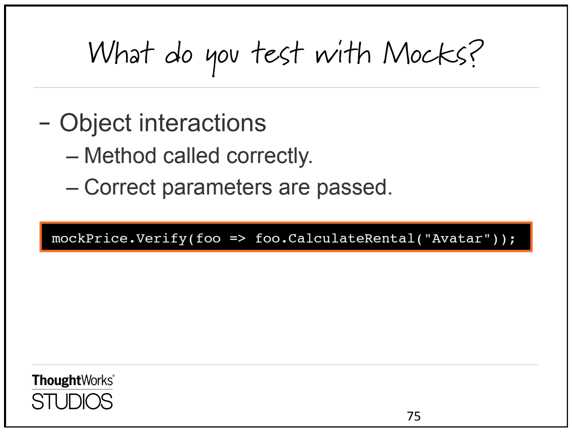
How do you use it (Moq)?

- Object interactions
  - Method called correctly.
  - Correct parameters are passed.

```
var mockPrice = new Mock<IPrice>();
mockPrice.Setup(foo =>
foo.CalculateRental(It.IsAny<string>())).Returns(10);
```

ThoughtWorks STUDIOS

74



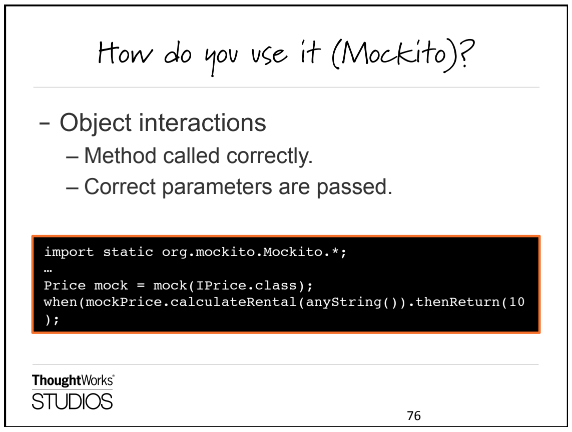
What do you test with Mocks?

- Object interactions
  - Method called correctly.
  - Correct parameters are passed.

```
mockPrice.Verify(foo => foo.CalculateRental("Avatar"));
```

ThoughtWorks STUDIOS

75



How do you use it (Mockito)?

- Object interactions
  - Method called correctly.
  - Correct parameters are passed.

```
import static org.mockito.Mockito.*;
…
Price mock = mock(IPrice.class);
when(mockPrice.calculateRental(anyString()).thenReturn(10
);
```

ThoughtWorks STUDIOS

76

What do you test?

- Object interactions
  - Method called correctly.
  - Correct parameters are passed.

```
verify(mockPrice).calculateRental("Avatar");
```

ThoughtWorks STUDIOS

77

Suggestions

- Mock/Stub instead of creating 'new' objects
- Only mock those methods that SUT calls
- Consider carefully where to use Mocks and where Stubs

ThoughtWorks STUDIOS

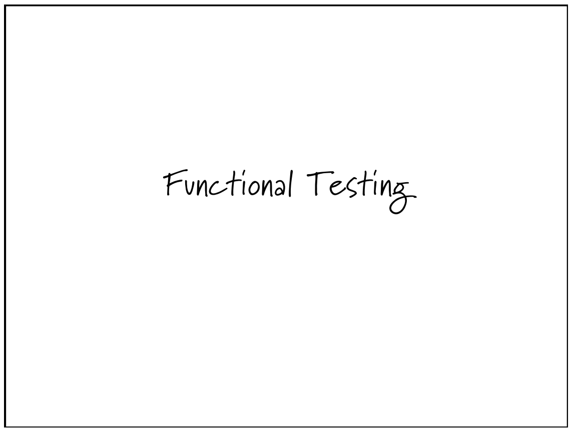
79

Mocking Exercise

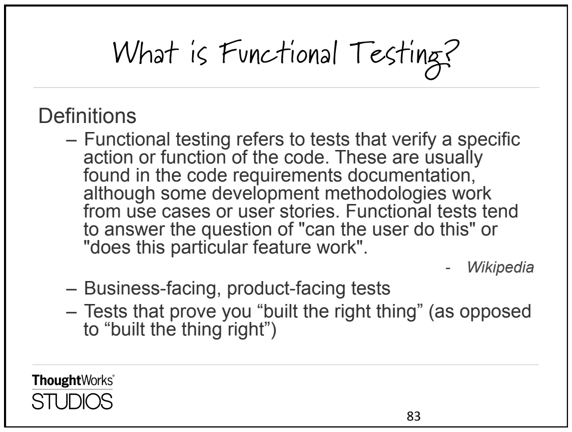
- Test the total ViewCurrentRentalsAction Mocking the customer or the transactions

ThoughtWorks STUDIOS

81



Functional Testing



What is Functional Testing?

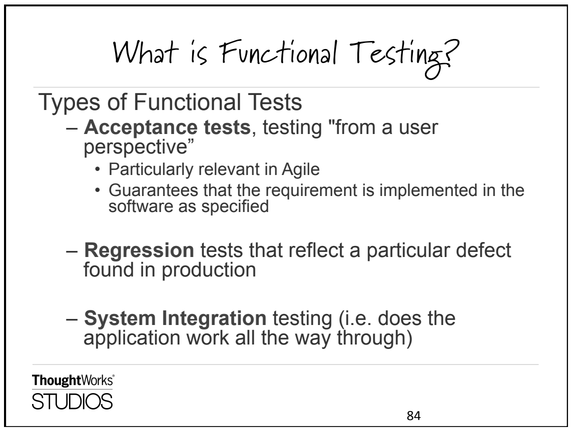
Definitions

- Functional testing refers to tests that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work".

  - *Wikipedia*
- Business-facing, product-facing tests
- Tests that prove you “built the right thing” (as opposed to “built the thing right”)

ThoughtWorks STUDIOS

83



What is Functional Testing?

Types of Functional Tests

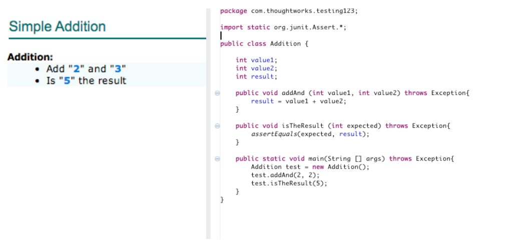
- **Acceptance tests**, testing "from a user perspective”
  - Particularly relevant in Agile
  - Guarantees that the requirement is implemented in the software as specified
- **Regression** tests that reflect a particular defect found in production
- **System Integration** testing (i.e. does the application work all the way through)

ThoughtWorks STUDIOS

84

What is Functional Testing?

Example of automated functional test



Simple Addition

Addition:
- Add "2" and "3"
- Is "5" the result

```
package com.thoughtworks.testing123;

import static org.junit.Assert.*;

public class Addition {

    int value1;
    int value2;
    int result;

    public void addAnd (int value1, int value2) throws Exception{
        result = value1 + value2;
    }

    public void isTheResult (int expected) throws Exception{
        assertEquals(expected, result);
    }

    public static void main(String [] args) throws Exception{
        Addition test = new Addition();
        test.addAnd(2, 2);
        test.isTheResult(5);
    }
}
```

85

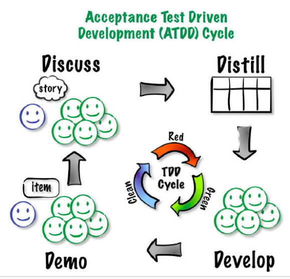
Functional Testing

Automating Functional Tests
- Functional Test Automation Makes Testing an Asset
- Living documentation
- Why Functional Test Automation?
- Unit Testing is Fundamentally different
- Automated Testing Has Additional "Gotchas"

ThoughtWorks STUDIOS

86

ATDD Functional Testing



ThoughtWorks STUDIOS

87

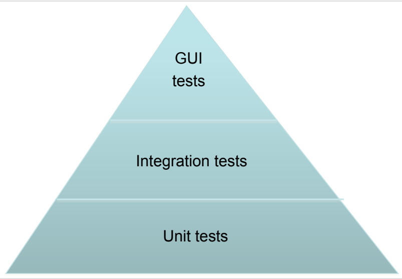
Functional Testing

Strategy
- How much?
- What should we automate vs. leave manual?
- Which tools?
- When in the development cycle?
- Who does the automation?
- What mix of testing makes sense?

ThoughtWorks STUDIOS

88

Functional Testing



ThoughtWorks STUDIOS

89

Story #2

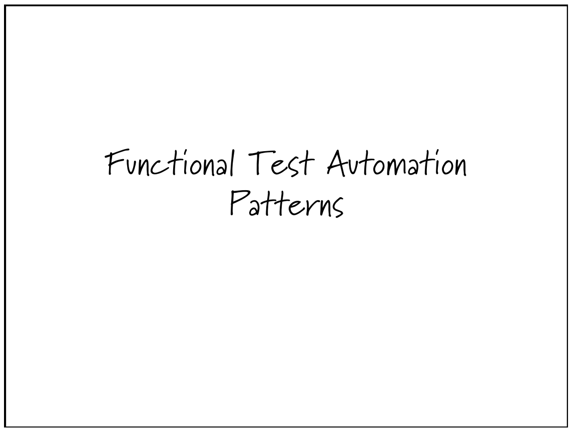
As a content provider

I want to have a promotion where every three-day rental gets an extra day for children's releases

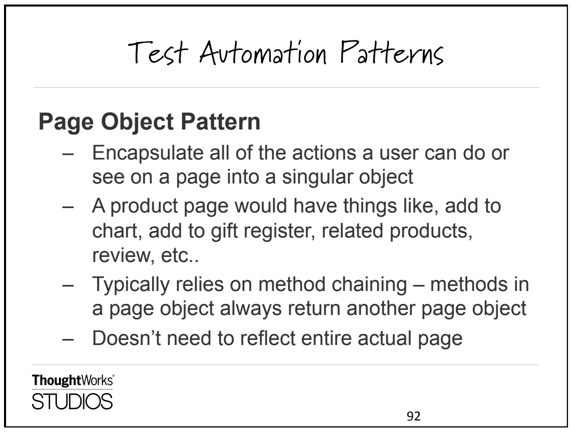
So that I can encourage longer rentals

ThoughtWorks STUDIOS

90



Functional Test Automation Patterns



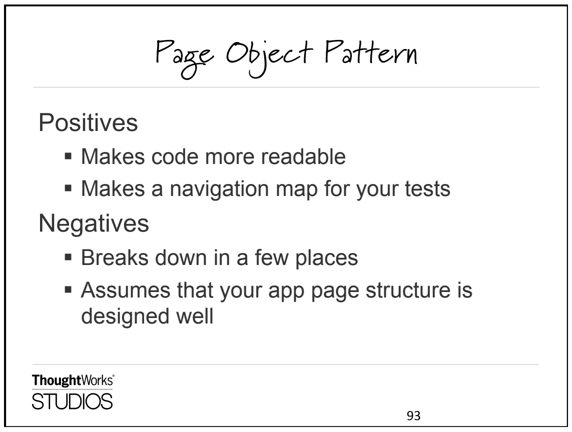
Test Automation Patterns

Page Object Pattern

- Encapsulate all of the actions a user can do or see on a page into a singular object
- A product page would have things like, add to chart, add to gift register, related products, review, etc..
- Typically relies on method chaining – methods in a page object always return another page object
- Doesn’t need to reflect entire actual page

ThoughtWorks STUDIOS

92



Page Object Pattern

Positives

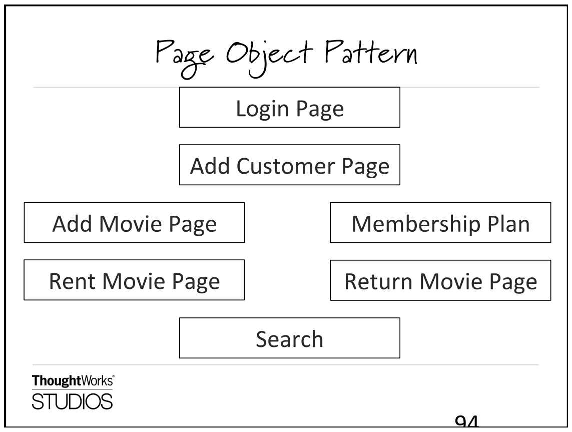
- Makes code more readable
- Makes a navigation map for your tests

Negatives

- Breaks down in a few places
- Assumes that your app page structure is designed well

ThoughtWorks STUDIOS

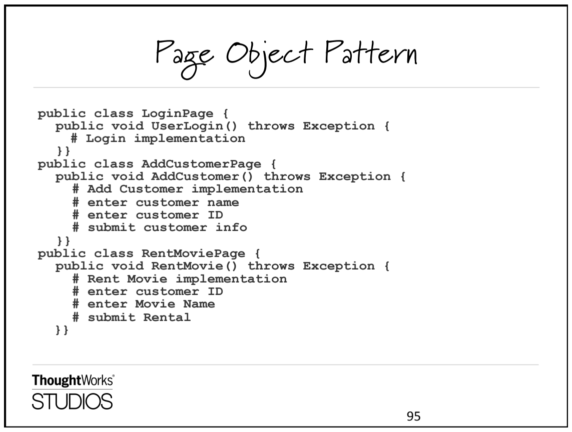
93



Page Object Pattern

ThoughtWorks STUDIOS

94

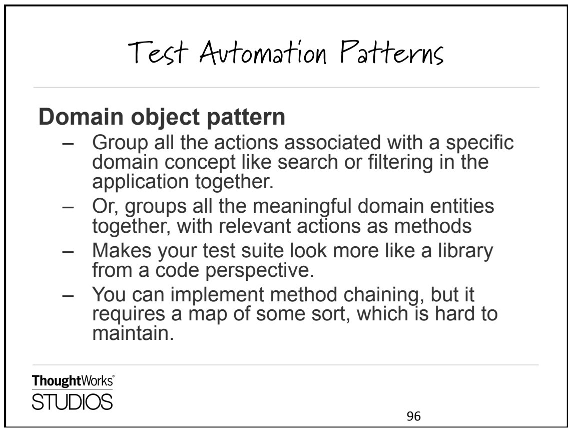


Page Object Pattern

```
public class LoginPage {
  public void UserLogin() throws Exception {
    # Login implementation
  }}
public class AddCustomerPage {
  public void AddCustomer() throws Exception {
    # Add Customer implementation
    # enter customer name
    # enter customer ID
    # submit customer info
  }}
public class RentMoviePage {
  public void RentMovie() throws Exception {
    # Rent Movie implementation
    # enter customer ID
    # enter Movie Name
    # submit Rental
  }}
```

ThoughtWorks STUDIOS

95



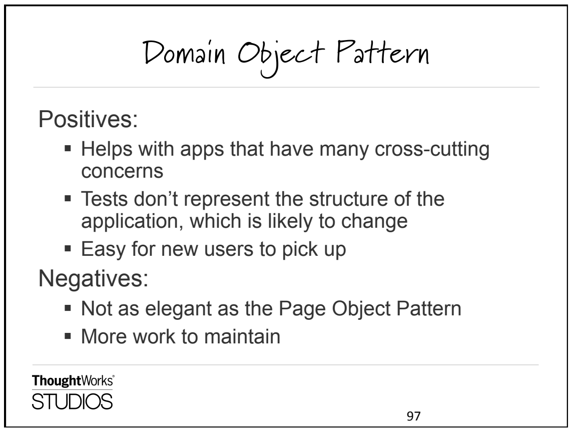
Test Automation Patterns

Domain object pattern

- Group all the actions associated with a specific domain concept like search or filtering in the application together.
- Or, groups all the meaningful domain entities together, with relevant actions as methods
- Makes your test suite look more like a library from a code perspective.
- You can implement method chaining, but it requires a map of some sort, which is hard to maintain.

ThoughtWorks STUDIOS

96



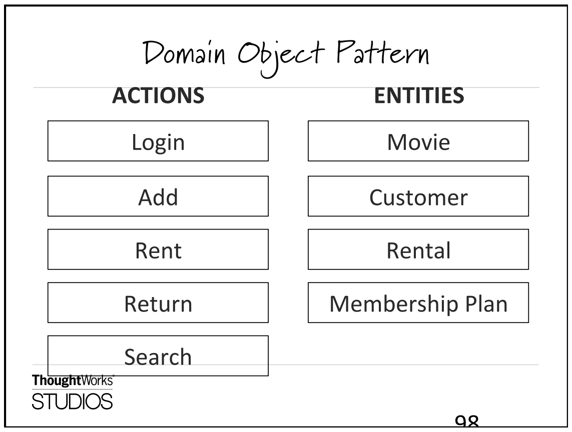
Domain Object Pattern

Positives:
- Helps with apps that have many cross-cutting concerns
- Tests don’t represent the structure of the application, which is likely to change
- Easy for new users to pick up

Negatives:
- Not as elegant as the Page Object Pattern
- More work to maintain

ThoughtWorks STUDIOS

97

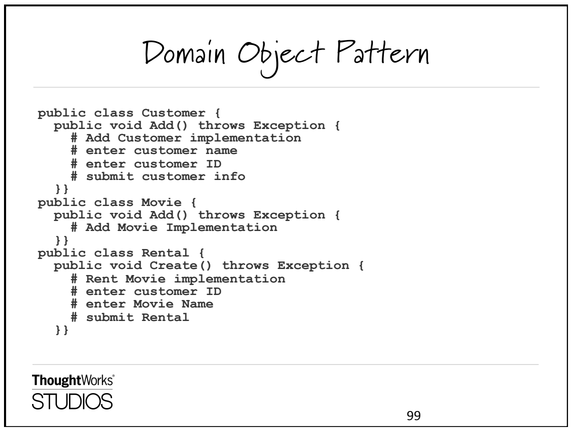


Domain Object Pattern

ACTIONS	ENTITIES
Login	Movie
Add	Customer
Rent	Rental
Return	Membership Plan
Search	

ThoughtWorks STUDIOS

98



Domain Object Pattern

```
public class Customer {
  public void Add() throws Exception {
    # Add Customer implementation
    # enter customer name
    # enter customer ID
    # submit customer info
  }}
public class Movie {
  public void Add() throws Exception {
    # Add Movie Implementation
  }}
public class Rental {
  public void Create() throws Exception {
    # Rent Movie implementation
    # enter customer ID
    # enter Movie Name
    # submit Rental
  }}
```

ThoughtWorks STUDIOS

99

Functional Testing

RECOMMENDED PRACTICES

100

Recommended Practices

UI Changes
- Xpath is difficult to maintain
  Avoid using xpaths
- ID change
  Create constant IDs & reuse

Follow DRY principle

ThoughtWorks STUDIOS

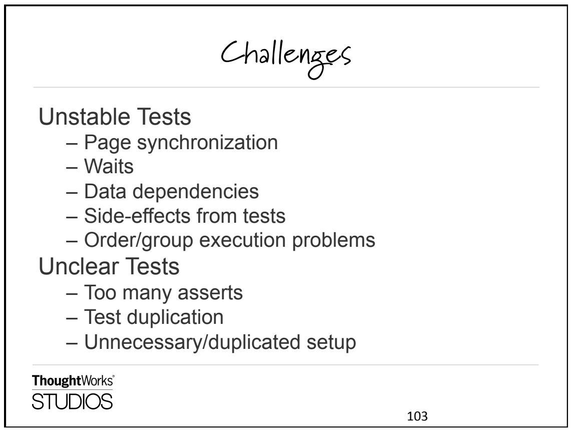
101

Recommended Practices

Scenarios
- Write from business perspective
- Write tests for reusability
- Refactor steps
- Extract steps as you go
- Rule of Thumb for workflow – not more than 10 steps
- Limit verifications and assertions in extracted steps

ThoughtWorks STUDIOS

102



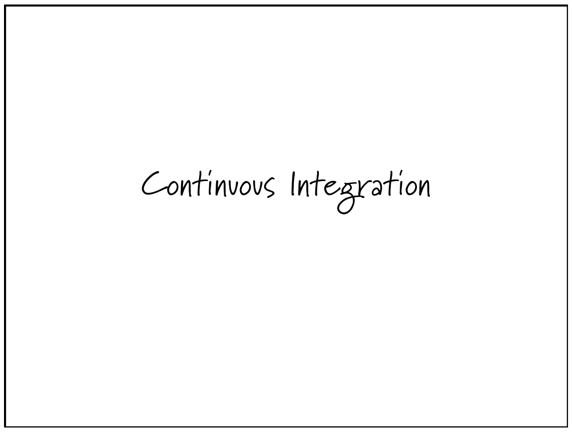
Challenges

Unstable Tests
- Page synchronization
- Waits
- Data dependencies
- Side-effects from tests
- Order/group execution problems

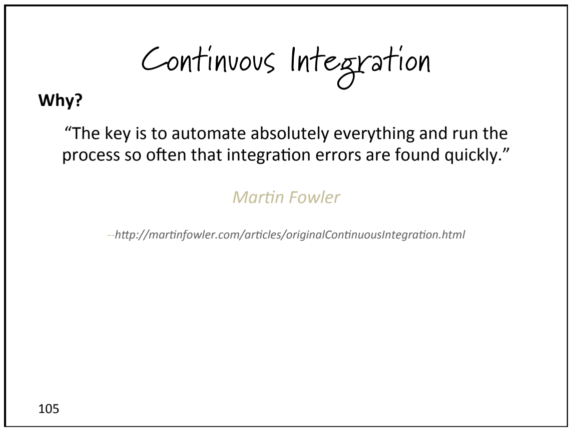
Unclear Tests
- Too many asserts
- Test duplication
- Unnecessary/duplicated setup

ThoughtWorks STUDIOS

103



Continuous Integration



Continuous Integration

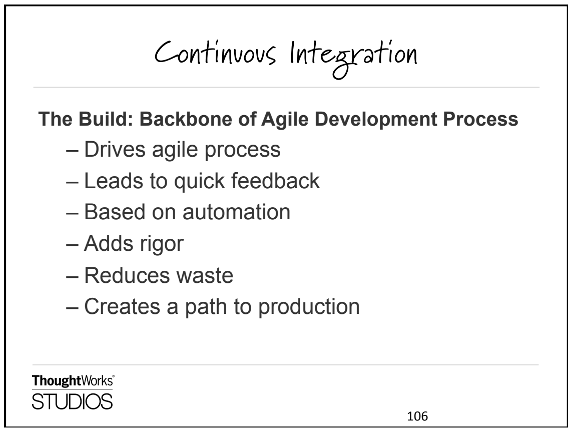
Why?

"The key is to automate absolutely everything and run the process so often that integration errors are found quickly."

Martin Fowler

--http://martinfowler.com/articles/originalContinuousIntegration.html

105



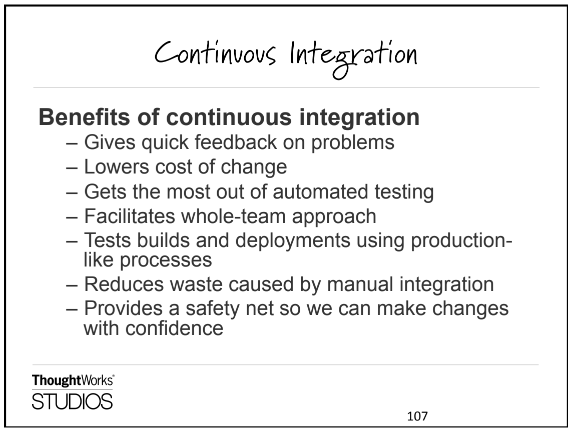
Continuous Integration

The Build: Backbone of Agile Development Process

– Drives agile process
– Leads to quick feedback
– Based on automation
– Adds rigor
– Reduces waste
– Creates a path to production

ThoughtWorks STUDIOS

106



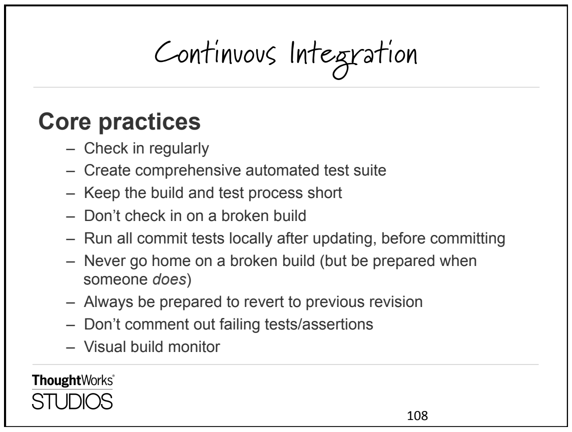
Continuous Integration

Benefits of continuous integration

– Gives quick feedback on problems
– Lowers cost of change
– Gets the most out of automated testing
– Facilitates whole-team approach
– Tests builds and deployments using production-like processes
– Reduces waste caused by manual integration
– Provides a safety net so we can make changes with confidence

ThoughtWorks STUDIOS

107



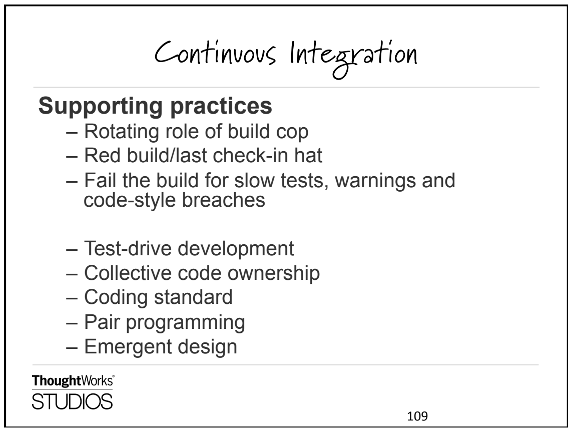
Continuous Integration

Core practices

– Check in regularly
– Create comprehensive automated test suite
– Keep the build and test process short
– Don't check in on a broken build
– Run all commit tests locally after updating, before committing
– Never go home on a broken build (but be prepared when someone *does*)
– Always be prepared to revert to previous revision
– Don't comment out failing tests/assertions
– Visual build monitor

ThoughtWorks STUDIOS

108



Continuous Integration

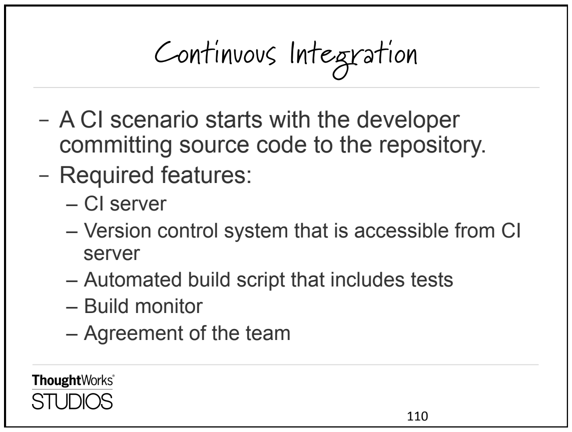
Supporting practices

- Rotating role of build cop
- Red build/last check-in hat
- Fail the build for slow tests, warnings and code-style breaches

- Test-drive development
- Collective code ownership
- Coding standard
- Pair programming
- Emergent design

ThoughtWorks STUDIOS

109



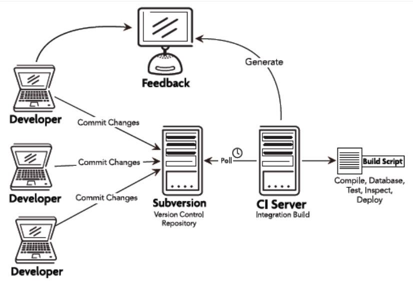
Continuous Integration

- A CI scenario starts with the developer committing source code to the repository.
- Required features:
  - CI server
  - Version control system that is accessible from CI server
  - Automated build script that includes tests
  - Build monitor
  - Agreement of the team

ThoughtWorks STUDIOS

110

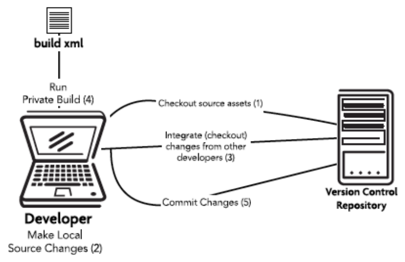
Continuous Integration



ThoughtWorks STUDIOS

111

Workflow/Check-in steps


build xml
Run
Private Build (4)
Checkout source assets (1)
Integrate (checkout) changes from other developers (3)
Commit Changes (5)
Version Control Repository
Developer
Make Local Source Changes (2)

ThoughtWorks STUDIOS

112

What is a successful build?

- All the latest sources are checked out of the configuration management system
- Every file is compiled from scratch
- The resulting compiled code is put into binaries
- The system is started and suite of tests is run against the system

ThoughtWorks STUDIOS

113

Continuous Integration

Smells

- Too many red builds
- Long-running builds

ThoughtWorks STUDIOS

114

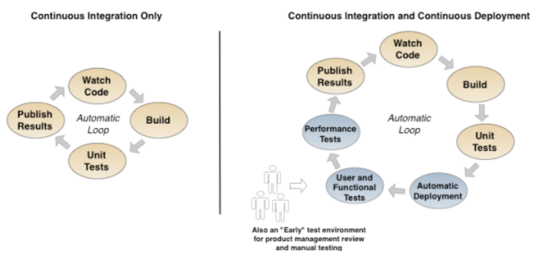
Other considerations

- Merging is a nightmare – why?
- To branch or not to branch?
- Definition of done
- Code metrics

ThoughtWorks STUDIOS

115

Continuous Integration and Deployment


Continuous Integration Only
Watch Code
Build
Unit Tests
Publish Results
Automatic Loop
Continuous Integration and Continuous Deployment
Watch Code
Build
Unit Tests
Automatic Deployment
User and Functional Tests
Performance Tests
Publish Results
Automatic Loop
Also an "Early" test environment for product management review and manual testing

ThoughtWorks STUDIOS

116

Story #8

As a sales manager

I want to allow a free regular movie when the customer has 5 frequent renter points

So that I can encourage repeat customer

ThoughtWorks STUDIOS

117

Continuous Delivery

Continuous Delivery Definition

Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.

Martin Fowler

ThoughtWorks STUDIOS

119

Continuous Delivery value proposition

Main benefits

– Increasing the throughput of new features and also the stability of the production systems: increasing innovation and predictability together at the same time

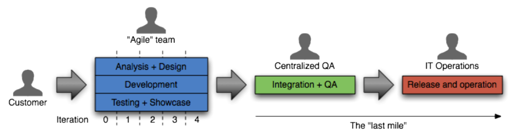
– Close the gap between the Business and IT, between the Customers, the Market and the Business

ThoughtWorks STUDIOS

120

Continuous Delivery

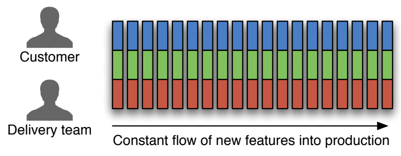
Traditional approach to delivery



121

Continuous Delivery

The ideal approach to delivery



122

Continuous Delivery

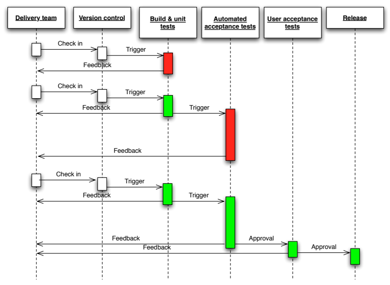
Principles

- Create a repeatable process for releasing software
- Automate almost everything
- Keep everything in version control
- If it hurts, do it more often, and bring the pain forward
- Build quality in (Deming)
- Done means released
- Everybody is responsible for delivery
- Continuous improvement

ThoughtWorks STUDIOS

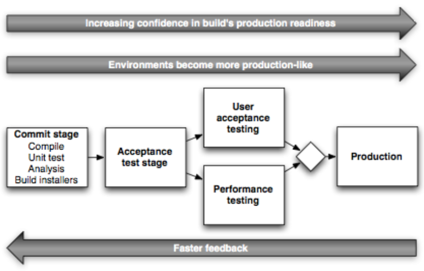
123

Deployment Pipeline



124

Deployment Pipeline



125

Continuous Delivery

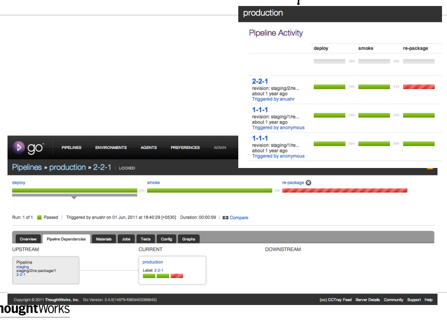
Stages and environments

- Commit stage
- Acceptance stage
- Manual stages
- Deployment stages

ThoughtWorks STUDIOS

126

Continvous Delivery with Go



2-2-1
revision: staging/2/re...
about 1 year ago
Triggered by anushr

1-1-1
revision: staging/1/re...
about 1 year ago
Triggered by anonymous

1-1-1
revision: staging/1/re...
about 1 year ago
Triggered by anonymous

Run: 1 of 1 | Passed | Triggered by anushr on 01 Jun, 2011 at 18:40:29 [+0530] Duration: 00:00:59 | Compare

Pipeline
staging
staging/2/re-package/1
2-2-1

production
Label: 2-2-1

Copyright © 2011 ThoughtWorks, Inc. Go Version: 2.4.0(14979-f080d403366645)

ThoughtWorks STUDIOS

127

Continvous Delivery

Guidelines

– Build your binaries only *once*
– Deploy the same way to *every environment*
– Smoke test your deployments
– Keep your environments similar
– If anything fails, stop the line

ThoughtWorks STUDIOS

128

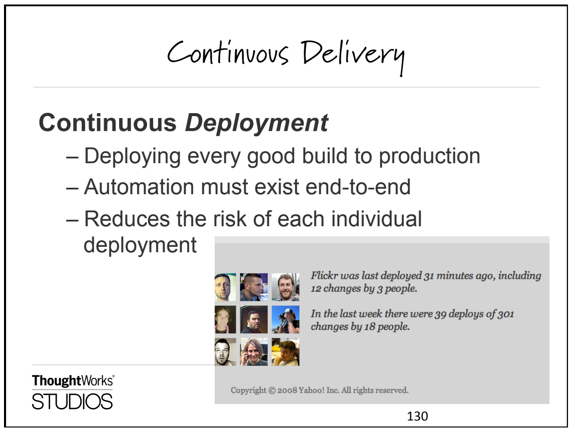
Continvous Delivery

Practices

– Automated testing and deploy, C.I.
– Trunk Based Development
– Hot deploy with zero-downtime
– Infrastructure automation
– Monitoring and logging

ThoughtWorks STUDIOS

129



Continuous Delivery

Continuous *Deployment*

- Deploying every good build to production
- Automation must exist end-to-end
- Reduces the risk of each individual deployment

Flickr was last deployed 31 minutes ago, including 12 changes by 3 people.

In the last week there were 39 deploys of 301 changes by 18 people.

Copyright © 2008 Yahoo! Inc. All rights reserved.

ThoughtWorks STUDIOS

130

ThoughtWorks STUDIOS

mingle twist go

Agile Development Practices

Thank you!