

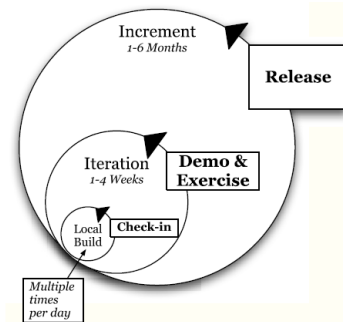
ACCEPTANCE TESTING APPLICATION BEHAVIOR

Abstract

- * How do you ensure your applications meet the expectations of your key customers? In this session we will explore using the FIT tool and Behavior Driven Design tools to do exactly this.
- * Unit Testing helps you, the programmer, verify your application meets and continues to meet your expectations. But how do you ensure that your application meets and continues to meet the real expectations, those of your domain experts and key customers? We will take a look at two distinct approaches for customer acceptance testing. We will take a look at using FIT for testing behavior and at Behavior Driven Design tools and techniques. Both these approaches can help us create what is called executable documentation and to stay sane in the world of changing requirements and evolutionary design.

Agile Development

- * What's Agility?
- * It's all about
 - ☆ Building relevant working software
 - ☆ By constantly getting feedback

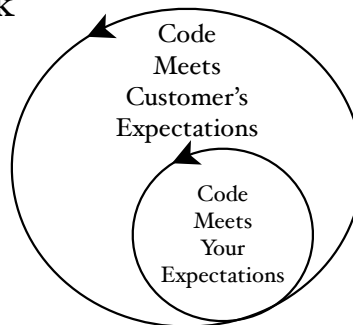


"Feeling the Rhythm" —Practices of an Agile Developer

3

Essence of Agility

- * To create relevant working software
- * Developing software is hard business
- * How can you succeed?
- * Feedback is essential
- * Two kinds of feedback



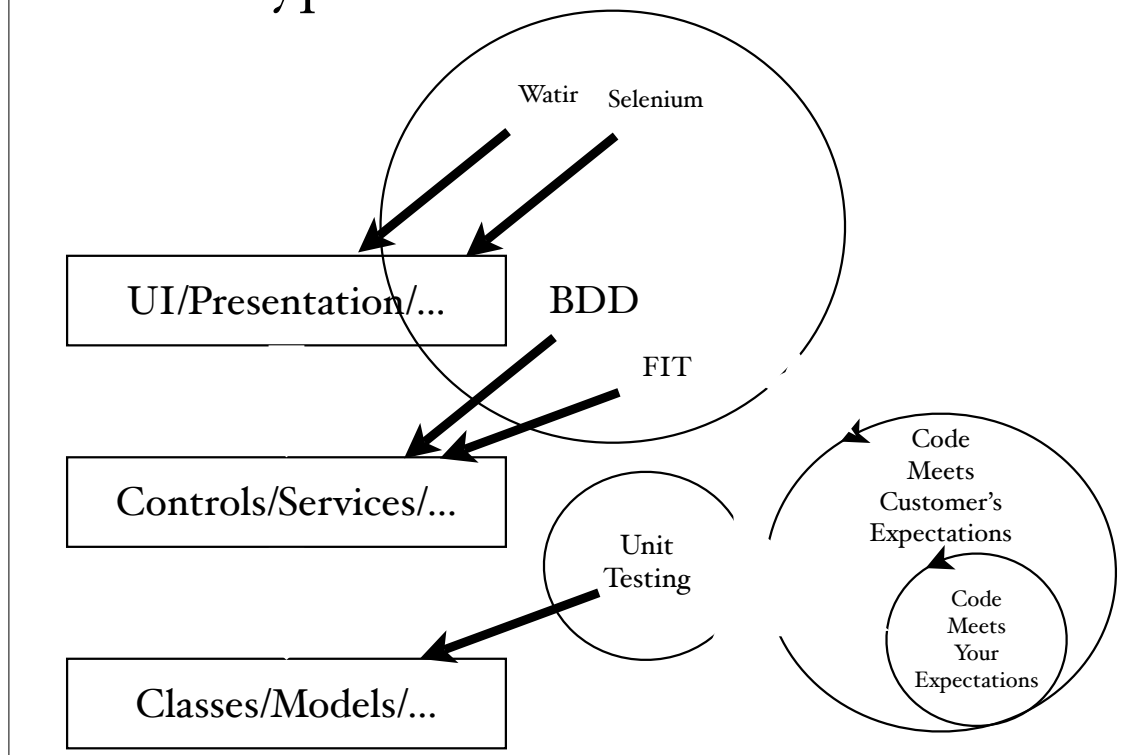
4

A Key Ingredient

- * Testing is a key ingredient in Agile development
- * Black box testing
 - * Tester does not know or does not care implementation and design details
- * White box testing
 - * Tester is aware of and is interested in design/implementation details
 - * Unit Testing falls under this

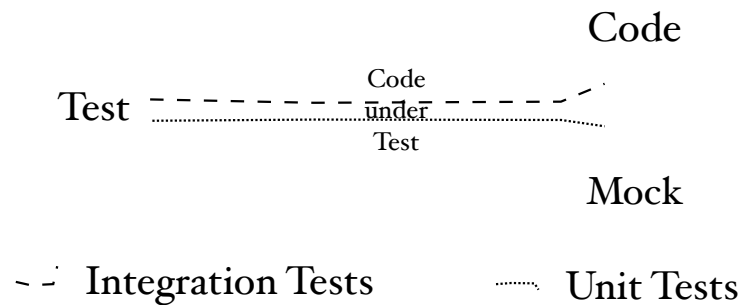
5

Types of Tests and Levels



Unit Vs. Integration Tests

- * Unit Tests often carried out in isolation on a unit of code without its dependencies (or mocking those)
- * Integration tests often carried out in full mode of operation of the functionality with system dependencies



7

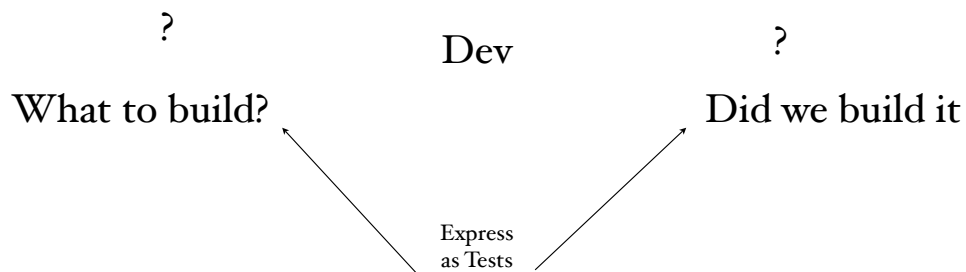
Unit Test Not Sufficient

- * Unit Testing is essential, but not sufficient
- * Unit Tests help assert code meets developer's expectations
- * How do you know if that meets users' expectations?
- * It's the difference between building software right and building right software

8

Gathering Requirements

- * Capturing requirements is a challenge
- * We've seen several approaches
- * Use Case helps, but often tends to be heavy weight and in effective beyond certain point of diminishing returns
- * Agile Developers use User Stories



3 Cs of User Stories

- * When creating User Stories, you focus on 3 Cs
- * Card
 - * Feature expressed in an index card
- * Conversation
 - * Use short description as starting point for useful discussions that promote exploration and understanding
- * Confirmation
 - * Helps you know when you're done—tests are written as a way to confirm completion of feature development

INVEST in User Stories

* *I*ndependent

● *E*stimable

* *N*egotiable

● *S*mall

* *V*aluable

II

"User Stories Applied: For Agile Software Development"—Mike Cohn

Expressing Requirements

- Writing tests help communicate the “What’s” of an application by way of concrete examples
- It helps to make sure the application is doing what’s expected
- Keeps an eye on it as system continues to evolve

I2

Ubiquitous Language

- Domain Driven Design emphasizes focus on Domain/
Business
- Ubiquitous Language is used as a means of communication
among domain experts, developers, and between the two
groups
- Tests serve as an Ubiquitous language that promotes such
communication in a precise manner

13

Fit

- Framework for Integration Testing
 - "General purpose open-ended framework for expressing
tests"—Developed in 2002 by Ward Cunningham
- Helps focus on business perspectives
- Tables represent tests—easy for non-programmers to use
- Automated checking and reporting of results
- Useful for
 - Business Rules related to Business Calculations
 - Business Rules related to Business Process/Workflow

14

Why Fit?

- Promotes communication
- Precise way to express expectations
- Helps know when you're done
- Keeps an eye on it as system evolves
- Instant alert when things fall apart

15

Who's fit for (F)it?

- Business Analyst
- Testers
- Developers
 - Architects, team leads, programmers,...

16

Strength in Simplicity

- Simple tables to express examples of expectations
- Easy for just about anyone to use
- Fosters communication

17

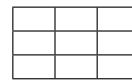
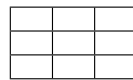
Strength in Simplicity

- Simple tables to express examples of expectations
- Easy for just about anyone to use
- Fosters communication

18

How does it fit together?

- Table of Tests
 - Expresses expectations by way of examples
- Fixture
 - Checks that the system satisfied the given tests



Order of tests may be important to reader, but Fit treats them independent of each other (except for ActionFixture)

Fixture

Fixture

System

19

Fixtures

- Column Fixture
 - Helps test calculations
- Action Fixture
 - Helps test events or actions
- Row Fixture
 - Helps test collections

20

Table

FixtureName						
given1	given2	given3	...	calculated1	calculated2	...

21

Test Results

Passed

Failed—more info provided

Part of test not complete or something messed up

Part of table not processed (ignored)

22

ColumnFixture

- Useful to check business logic that calculates something
- You will have to isolate the business logic in order to test it
 - That's a good thing!

23

Creating Table & Fixture

com.agiledeveloper.fixture.PriceFixture	
product	getPrice()
Pepsi	100
Diet-Pepsi	100
Twist	80
Lemon	60
Crystal	70
Coke	error

price.html

```
package com.agiledeveloper.fixture;
import fit.ColumnFixture;
public class PriceFixture extends ColumnFixture
{
    public String product;
    public int getPrice()
    {
        return 0;
    }
}
```

Fixture: Glue code

24

Running FIT

```
> grep fit ~/.bash_login
alias fit="java -classpath /usr/local/bin/fit/fit.jar:../../FitExample.jar:. fit.FileRunner"
> fit price.html report.html
0 right, 6 wrong, 0 ignored, 0 exceptions
> open report.html
```

com.agiledeveloper.fixture.PriceFixture	
product	getPrice()
Pepsi	100 <i>expected</i>
	0 <i>actual</i>
Diet-Pepsi	100 <i>expected</i>
	0 <i>actual</i>
Twist	80 <i>expected</i>
	0 <i>actual</i>
Lemon	60 <i>expected</i>
	0 <i>actual</i>
Crystal	70 <i>expected</i>
	0 <i>actual</i>
Coke	error <i>expected</i>
	0 <i>actual</i>

25

Fixing Code

```
package com.agiledeveloper.fixture;

import fit.ColumnFixture;
import com.agiledeveloper.VendingMachine;

public class PriceFixture extends ColumnFixture
{
    public String product;
    private VendingMachine _vendingMachine = new VendingMachine();

    public int getPrice()
    {
        return _vendingMachine.getPrice(product);
    }
}
```

26

Fixing Code...

```
package com.agiledeveloper;
import java.util.Hashtable;

public class VendingMachine
{
    private Hashtable<String, Integer> _price =
        new Hashtable<String, Integer>();

    public VendingMachine()
    {
        _price.put("Pepsi", 100);
        _price.put("Diet-Pepsi", 100);
        _price.put("Twist", 80);
        _price.put("Lemon", 60);
        _price.put("Crystal", 70);
    }

    public int getPrice(String product)
    {
        if (!_price.containsKey(product))
        {
            throw new IllegalArgumentException();
        }

        return _price.get(product);
    }
}
```

27

reFITing it

```
> fit price.html report.html
6 right, 0 wrong, 0 ignored, 0 exceptions
```

com.agiledeveloper.fixture.PriceFixture	
product	getPrice()
Pepsi	100
Diet-Pepsi	100
Twist	80
Lemon	60
Crystal	70
Coke	error

28

ActionFixture

- Tests the effect of one or more sequence of actions—Device to control tests
- First cell is a command to ActionFixture
- start
 - Specifies the Actor (derived from fit.Fixture) class to instantiate and send subsequent actions to until another start
- enter
 - Second cell specifies name of method, with one parameter (third cell), on actor
- press
 - Second cell specifies name of method to call on actor
- check
 - Second cell specifies name of method to call on actor
 - Third cell represents expected value from this call

29

Creating Table & Fixture

fit.ActionFixture		
start	com.agiledeveloper.fixture.CollectCoinsFixture	
check	balance	0
enter	deposit	25
check	balance	25
enter	deposit	25
check	balance	50
enter	deposit	25
check	balance	75
enter	deposit	25
check	balance	100
enter	deposit	25
check	balance	100
enter	product	Pepi
press	getSoda	
check	balance	0

collectCoin.html

30

Creating Table & Fixture

```
package com.agiledeveloper.fixture;
import com.agiledeveloper.VendingMachine;

public class CollectCoinsFixture extends fit.Fixture
{
    private VendingMachine _vendingMachine = new VendingMachine();
    private String _productName;

    public int balance()
    {
        return _vendingMachine.getBalance();
    }

    public void deposit(int amount)
    {
        _vendingMachine.insertCoin(amount);
    }

    public void product(String productName)
    {
        _productName = productName;
    }

    public void getSoda()
    {
        _vendingMachine.dispense(_productName);
    }
}
```

31

Fixing Code

```
package com.agiledeveloper;
import java.util.Hashtable;

public class VendingMachine
{
    private Hashtable<String, Integer> _price =
        new Hashtable<String, Integer>();
    private int _balance;

    public VendingMachine()
    {
        ...
    }

    public int getPrice(String product)
    {
        ...
    }

    public int getBalance()
    {
        return _balance;
    }

    public void insertCoin(int amount)
    {
        if (_balance < 100) _balance += amount;
    }

    public void dispense(String productName)
    {
        _balance = 0; // Mimimal code for now...
    }
}
```

32

FITting it

```
> fit collectCoin.html report.html
7 right, 0 wrong, 0 ignored, 0 exceptions
> open report.html
```

fit.ActionFixture		
start	com.agiledeveloper.fixture.CollectCoinsFixture	
check	balance	0
enter	deposit	25
check	balance	25
enter	deposit	25
check	balance	50
enter	deposit	25
check	balance	75
enter	deposit	25
check	balance	100
enter	deposit	25
check	balance	100
enter	product	Pepi
press	getSoda	
check	balance	0

33

RowFixture

- Tests results of a query is as expected
- Result is expected to be a list or collection
 - May be treated as ordered or unordered
- Rows together form a single group

34

Creating Table & Fixture

com.agiledeveloper.fixture.ProductListFixture	
name	size
Pepsi	12
Diet-Pepsi	12
Twist	8
Vanilla	10

productList.html

35

Creating Table & Fixture

```
package com.agiledeveloper.fixture;
import com.agiledeveloper.VendingMachine;

public class CollectCoinsFixture extends fit.Fixture
{
    private VendingMachine _vendingMachine = new VendingMachine();
    private String _productName;

    public int balance()
    {
        return _vendingMachine.getBalance();
    }

    public void deposit(int amount)
    {
        _vendingMachine.insertCoin(amount);
    }

    public void product(String productName)
    {
        _productName = productName;
    }

    public void getSoda()
    {
        _vendingMachine.dispense(_productName);
    }
}
```

36

Fixing Code

```
public void dispense(String productName)
{...}

public ArrayList getProducts()
{
    ArrayList products = new ArrayList();

    products.add(new Product("Pepsi", 12));
    products.add(new Product("Diet-Pepsi", 12));
    products.add(new Product("Twist", 8));
    products.add(new Product("Crystal", 10));
    return products;
}
```

VendingMachine.java

37

FITting it

```
> fit productList.html report.html
6 right, 2 wrong, 0 ignored, 0 exceptions
> open report.html
```

com.agiledeveloper.fixture.ProductListFixture	
name	size
Pepsi	12
Diet-Pepsi	12
Twist	8
Vanilla <i>missing</i>	10
Crystal <i>surplus</i>	10

38

Running Multiple Tests

- You can mix and run multiple tables of tests
- They may be of different types
- You may group tests into folders

39

Creating Table & Fixture

fit.ActionFixture		
start	com.agiledeveloper.fixture.PurchaseFixture	
com.agiledeveloper.fixture.ProductCountFixture		
product	count()	
Pepsi	100	
fit.ActionFixture		
check	balance	0
enter	deposit	25
check	balance	25
enter	deposit	50
check	balance	75
enter	deposit	25
check	balance	100
enter	deposit	25
check	balance	100
enter	product	Pepsi
press	getSoda	
check	balance	0
com.agiledeveloper.fixture.ProductCountFixture		
product	count()	
Pepsi	99	

purchase.html

40

Creating Table & Fixture

```
package com.agiledeveloper.fixture;

import com.agiledeveloper.VendingMachine;

public class PurchaseFixture extends fit.Fixture
{
    public static VendingMachine _vendingMachine =
        new VendingMachine();
    private String _productName;

    public int balance()
    {
        return _vendingMachine.getBalance();
    }

    public void deposit(int amount)
    {
        _vendingMachine.insertCoin(amount);
    }

    public void product(String productName)
    {
        _productName = productName;
    }

    public void getSoda()
    {
        _vendingMachine.dispense(_productName);
    }
}
```

41

Creating Table & Fixture

```
package com.agiledeveloper.fixture;

import fit.ColumnFixture;

public class ProductCountFixture extends ColumnFixture
{
    public String product;

    public int count()
    {
        return PurchaseFixture._vendingMachine.getProductCount(product);
    }
}
```

42

Fixing Code

```
private int _pepsiCount = 100;
```

VendingMachine.java

```
public VendingMachine()
{...}

public int getPrice(String product)
{...}

public int getBalance()
{...}

public void insertCoin(int amount)
{...}

public void dispense(String productName)
{
    if (productName.equals("Pepsi")) { _pepsiCount--; }
    _balance = 0; // Minimal code for now...
}

public ArrayList getProducts()
{...}

public int getProductCount(String product)
{
    if (product.equals("Pepsi")) return _pepsiCount;
    return 0;
}
}
```

43

FITting it

```
> fit purchase.html report.html
8 right, 0 wrong, 0 ignored, 0 exceptions
> open report.html
```

fit.ActionFixture		
start	com.agiledeveloper.fixture.PurchaseFixture	
com.agiledeveloper.fixture.ProductCountFixture		
product	count()	
Pepsi	100	
fit.ActionFixture		
check	balance	0
enter	deposit	25
check	balance	25
enter	deposit	50
check	balance	75
enter	deposit	25
check	balance	100
enter	deposit	25
check	balance	100
enter	product	Pepsi
press	getSoda	
check	balance	0
com.agiledeveloper.fixture.ProductCountFixture		
product	count()	
Pepsi	99	

44

FitNesse

- In Fit, you've gotta create HTML table, run fit, open browser to view result
- Can it be easier?
- FitNesse provides a single web based UI for developing, running and viewing results of test
 - A Wiki based system
- Developed by Micah D. Martin and Robert C. Martin with contributions from a number of others

45

Starting Server

```
> run.sh -p 8080
FitNesse (20060719) Started...
  port:                8080
  root page:          fitness.wiki.FileSystemPage at ./FitNesseRoot
  logger:             none
  authenticator:      fitness.authentication.PromiscuousAuthenticator
  html page factory:  fitness.html.HtmlPageFactory
  page version expiration set to 14 days.
```

46

FitNesse Default Page



FrontPage IMPORTED

WELCOME TO FITNESSE!
THE FULLY INTEGRATED STANDALONE ACCEPTANCE TESTING FRAMEWORK AND WIKI.

Table of Contents	
A One-Minute Description	What is <i>FitNesse</i> ? Start here.
A Two-Minute Example	A brief example. Read this one next.
User Guide	Answer the rest of your questions here.
Acceptance Tests	<i>FitNesse's</i> suite of Acceptance Tests

[FrontPage] [RecentChanges]

Click to Edit Page to add Test

47

Add A Test Page



FrontPage
EDIT PAGE

```
!img-1 http://files/images/FitNesseLogoMedium.jpg
!1 Welcome to [[FitNesse]][FitNesse.FitNesse]!
!3 "The fully integrated standalone acceptance testing framework and wiki."

!c "'Table of Contents'"
!c [[A One-Minute Description][FitNesse.OneMinuteDescription]] "What is [[FitNesse]][FitNesse.FitNesse]? Start here."
!c [[A Two-Minute Example][FitNesse.TwoMinuteExample]] "A brief example. Read this one next."
!c [[User Guide][FitNesse.UserGuide]] "Answer the rest of your questions here."
!c [[Acceptance Tests][FitNesse.SuiteAcceptanceTests]] "FitNesse's suite of Acceptance Tests"


TestVendingMachinePrice
```

spreadsheet to FitNesse

Click to Edit Page

48

Add Test



TestVendingMachinePrice

EDIT PAGE


```
!path /usr/local/bin/fitnesse/fitnesse.jar
!path /Users/venkats/workarea/getFitExamples/FitExample.jar

! | com.agiledeveloper.fixture.PriceFixture |
product | getPrice() |
Pepsi | 100 |
Diet-Pepsi | 100 |
Twist | 80 |
Lemon | 60 |
Crystal | 70 |
Coke | error |
```

Save Spreadsheet to FitNesse FitNesse to Spreadsheet - Insert Fixture Table - ▾

49

Run Test




TestVendingMachinePrice

classpath: /usr/local/bin/fitnesse/fitnesse.jar
classpath: /Users/venkats/workarea/getFitExamples/FitExample.jar

com.agiledeveloper.fixture.PriceFixture	
product	getPrice()
Pepsi	100
Diet-Pepsi	100
Twist	80
Lemon	60
Crystal	70
Coke	error

[.FrontPage] [.RecentChanges]



TestVendingMachinePrice

TEST RESULTS

Tests Executed OK

Assertions: 6 right, 0 wrong, 0 ignored, 0 exceptions

classpath: /usr/local/bin/fitnesse/fitnesse.jar
classpath: /Users/venkats/workarea/getFitExamples/FitExample.jar

com.agiledeveloper.fixture.PriceFixture	
product	getPrice()
Pepsi	100
Diet-Pepsi	100
Twist	80
Lemon	60
Crystal	70
Coke	error

[.FrontPage] [.RecentChanges]

50

Behavior Driven Design

- * It is a TDD approach
- * It is a ubiquitous language
- * It is an executable documentation
- * It promotes communication
- * Helps develop common vocabulary and metaphor
- * Help you to get the "words" right
- * Can be used by programmers, testers, business analysts, domain experts, and customers.

51

Behavior and Story

- * You can use BDD to express Stories and Behaviors
- * Story Framework and Spec Framework
- * Stories correspond to User Stories—to express behavior at application level
- * Spec or Behavior correspond to expectations at class level—to express behavior at service/component level
- * These can help express requirements that can be specified, understood, and negotiated by developers, testers, business analysts, and business customers.

52

Behavior

- * Each behavior is expressed as a test/*exercise* method
- * It tells what the object *should* do
- * Notice the keyword "should"—that's a main focus in BDD—the *shoulds* and the *shouldn'ts*

53

Building Stories

- * You may define user stories as a series of acceptance criteria as scenarios

- * It has the givens, events, and outcomes
- * That is
 - * *Given* some initial condition(s),
 - * *When* event(s) occurs,
 - * *Then* ensure some outcome(s)

54

Tools for BDD

- * Java - JBehave, JDave, beanSpec, Instinct
- * Groovy - GSpect, easyb

55

easyb

- * Started by Andy Glover
- * Express Story and Spec using Groovy Based Domain Specific Language (DSL)
- * Highly expressive
- * Can be used for Java and Groovy applications

56

References

- * <http://fit.c2.com>
- * <http://www.fitnessse.org>
- * <http://behavior-driven.org>
- * <http://jbehave.org/>
- * <http://codeforfun.wordpress.com/gspec/>
- * <http://www.easyb.org/>

You can download examples and slides from
<http://www.agiledeveloper.com> - download