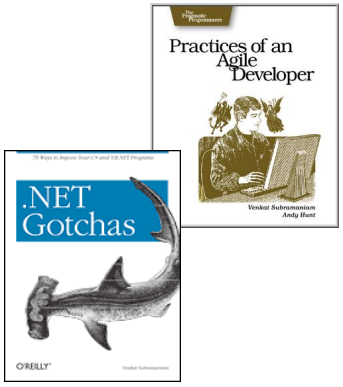
get Fit

spkr.name = 'Venkat Subramaniam'
spkr.company = 'Agile Developer, Inc.'
spkr.credentials = %w{Programmer Trainer Author}
spkr.blog = 'agiledeveloper.com/blog'
spkr.email = 'venkats@agiledeveloper.com'



Abstract

- Unit testing tells you, the programmer, that your code (and the change) meets your expectations. How do you know if you are meeting your customers' expectations? Agile development is all about feedback and doing what's relevant to the customers, isn't it? Framework for Integration testing or Fit helps you to automate tests for customer expectations.

- In this presentation we will learn how to write Fit tests and how to automate their execution. We will also use FitNesse.

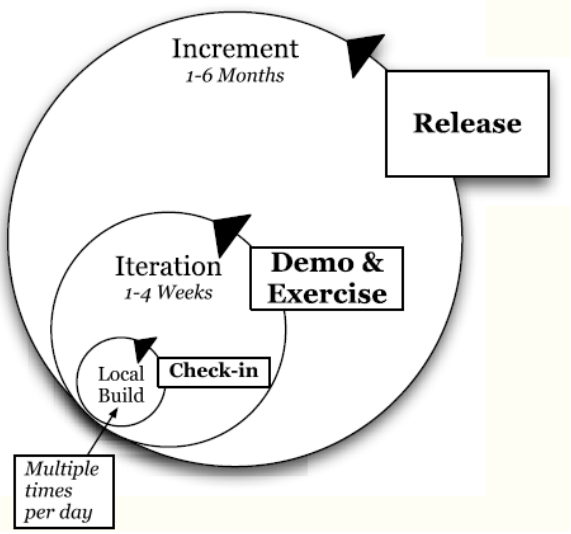
- Topics: Beyond Unit Testing, Integration Testing, Customer Expectations, Writing Fit Tests, Writing Fixtures, Automating tests, What is FitNesse, Using FitNesse

Agenda

- Agile Development
- Testing
- Requirements
- Fit
- FitNesse
- FitLibrary
- Guidelines
- Conclusion

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

A Key Ingredient

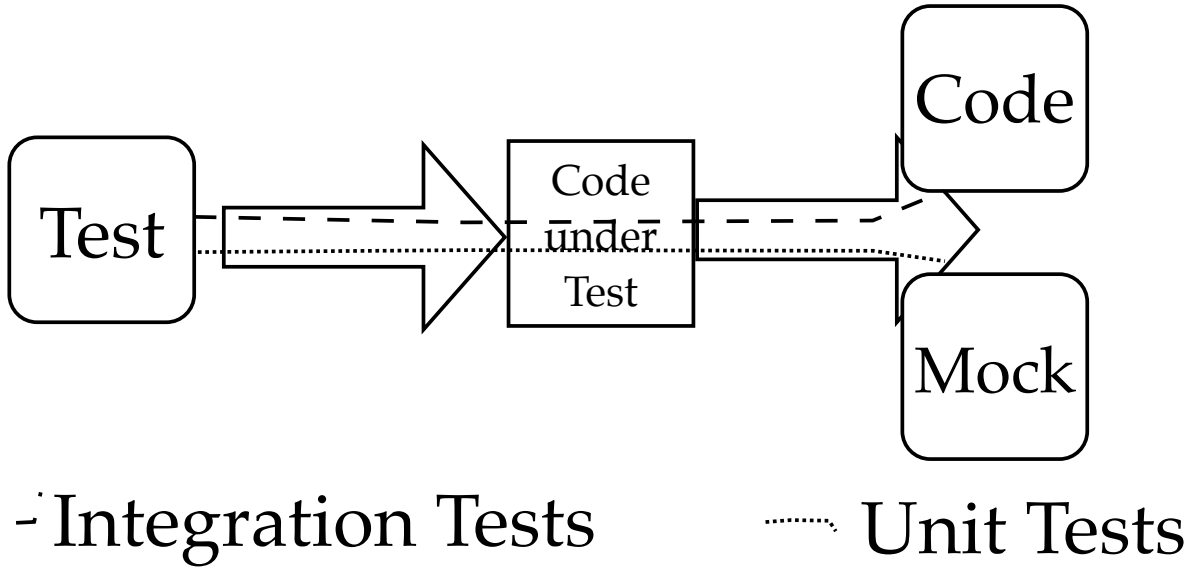
- Testing is a key ingredient in Agile development
- Black box testing
  - Tester does not know or does not care implementation and design details
- White box testing
  - Tester is aware of and is interested in design/ implementation details
  - Unit Testing falls under this

Agenda

- Agile Development
- Testing
- Requirements
- Fit
- FitNesse
- FitLibrary
- Guidelines
- Conclusion

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



Unit Test Not Sufficient

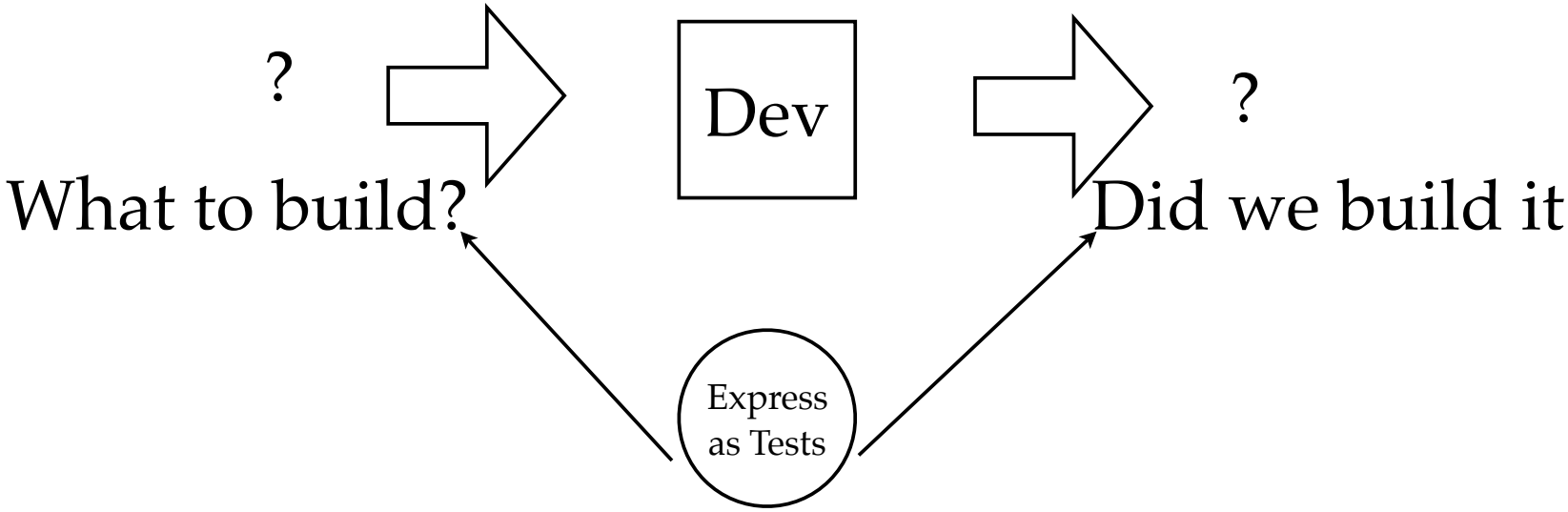
- Unit Testing is essential, but not sufficient
- Unit Tests help assert code meets developer's expectations
- How do you know if that meets users' expectations?
- It's the difference between building software right and building right software

Agenda

- Agile Development
- Testing
- Requirements
- Fit
- FitNesse
- FitLibrary
- Guidelines
- Conclusion

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



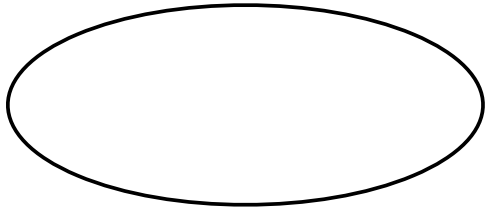
3 Cs of User Stories

- 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

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

INVEST in User Stories

- *I*ndependent
- *N*egotiable
- *V*aluable
- *E*stimable
- *S*mall



"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

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

Agenda

- Agile Development
- Testing
- Requirements
- Fit
- FitNesse
- FitLibrary
- Guidelines
- Conclusion

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

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

Who's fit for (F)it?

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

Strength in Simplicity

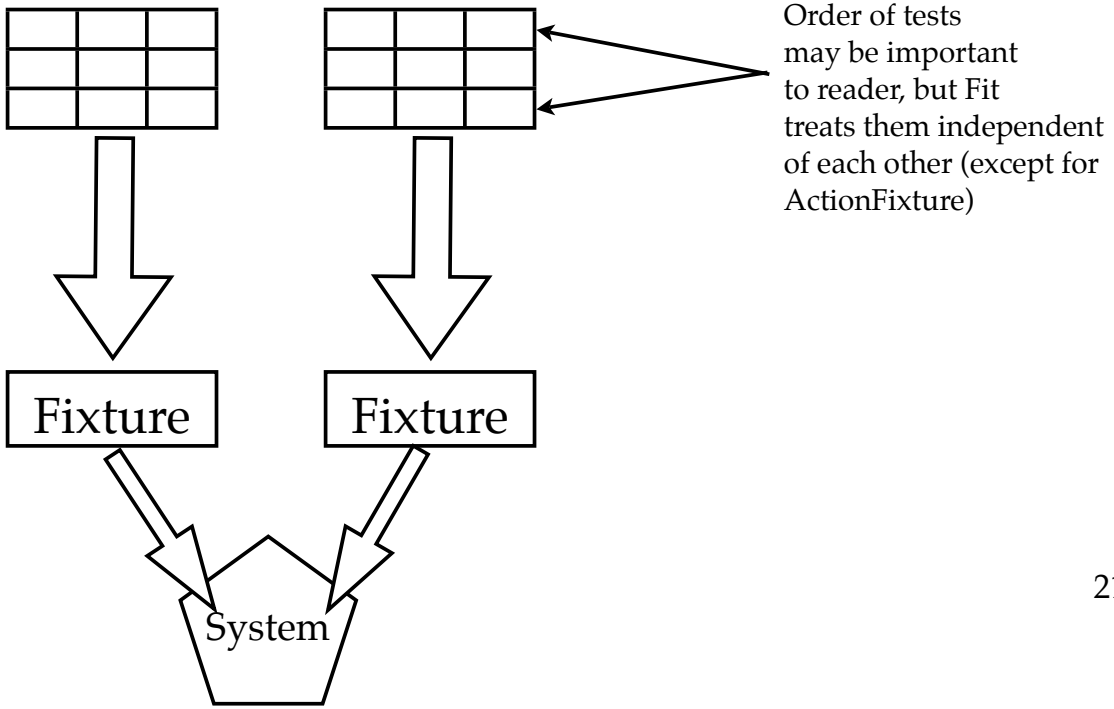
- Simple tables to express examples of expectations
- Easy for just about anyone to use
- Fosters communication

Strength in Simplicity

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

How does it fit together?

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



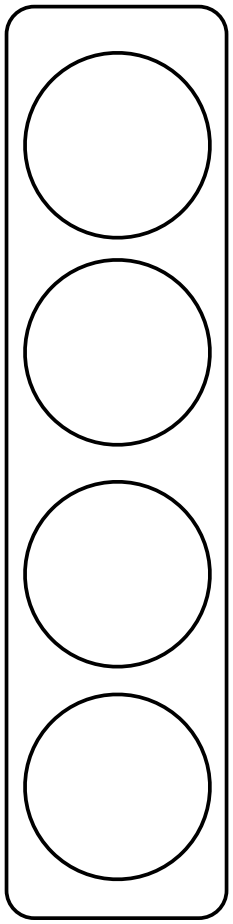
Fixtures

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

Table

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

Test Results



Passed

Failed—more info provided

Part of test not complete or something messed up

Part of table not processed (ignored)

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!

Creating Table & Fixture

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

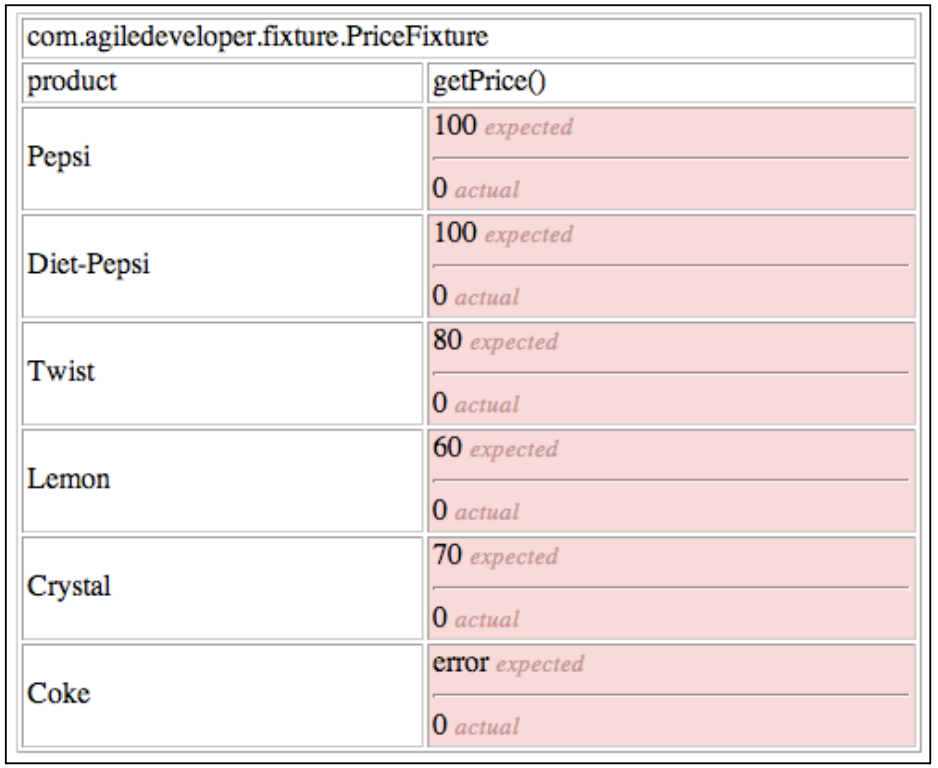
price.html

```
package com.agiledeveloper.fixture;

import fit.ColumnFixture;

public class PriceFixture extends ColumnFixture
{
    public String product;

    public int getPrice()
    {
        return 0;
    }
}
```

Fixture: Glue code

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 *expected* 0 *actual*
Diet-Pepsi	100 *expected* 0 *actual*
Twist	80 *expected* 0 *actual*
Lemon	60 *expected* 0 *actual*
Crystal	70 *expected* 0 *actual*
Coke	error *expected* 0 *actual*

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);
    }
}
```

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);
    }
}
```

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

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

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

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);
    }
}
```

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...
    }
}
```

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

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

Creating Table & Fixture

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

productList.html

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);
    }
}
```

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

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 *missing*	10
Crystal *surplus*	10

Running Multiple Tests

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

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

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);
    }
}
```

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);
    }
}
```

Fixing Code

VendingMachine.Java

```
    private int _pepsiCount = 100;

    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; // Mimimal code for now...
    }

    public ArrayList getProducts()
    {...}

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

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

Agenda

- Agile Development
- Testing
- Requirements
- Fit
- FitNesse
- FitLibrary
- Guidelines
- Conclusion

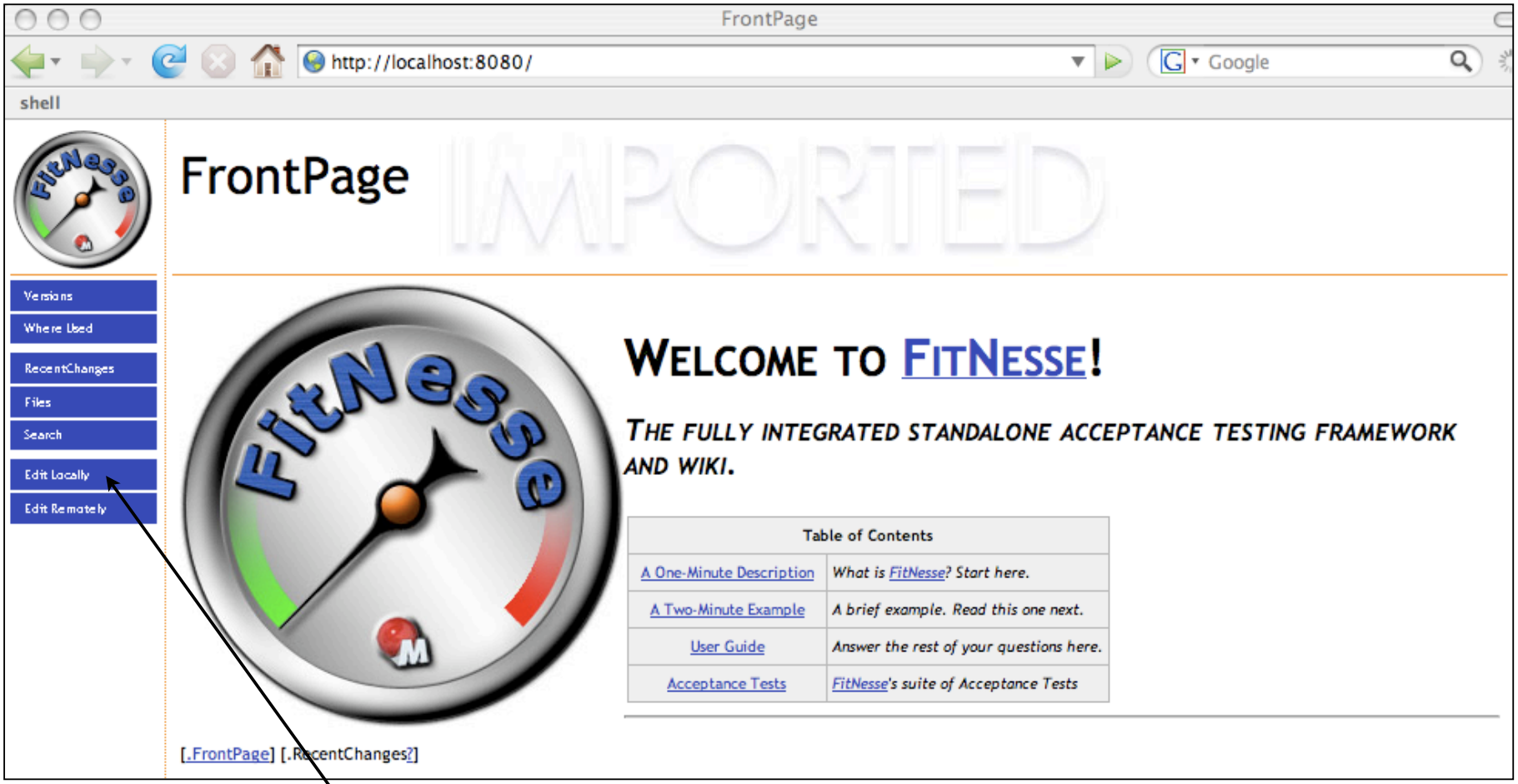
FitNesse

- 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

Starting Server

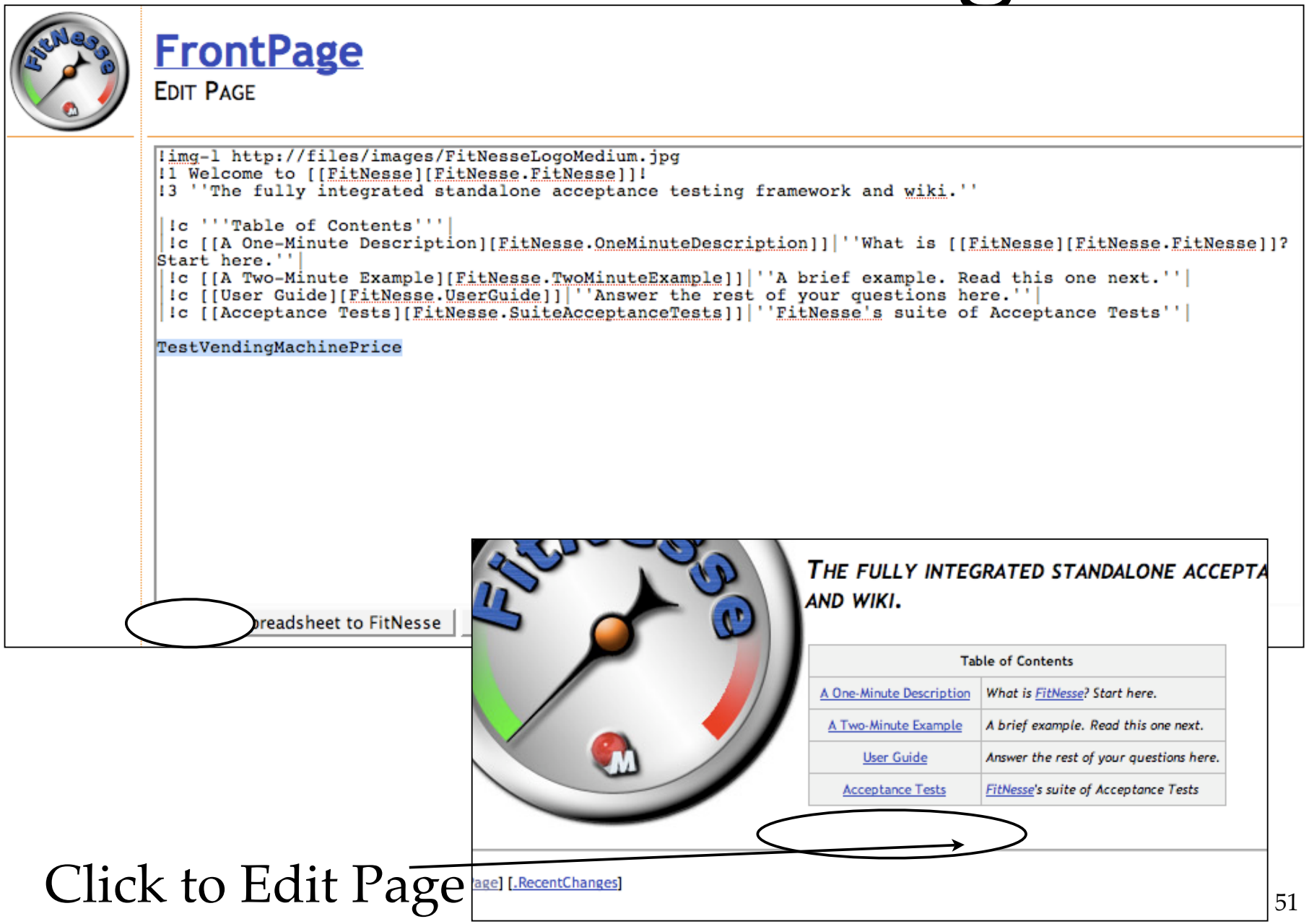
```
> run.sh -p 8080
FitNesse (20060719) Started...
        port:              8080
        root page:         fitnesse.wiki.FileSystemPage at ./FitNesseRoot
        logger:            none
        authenticator:     fitnesse.authentication.PromiscuousAuthenticator
        html page factory: fitnesse.html.HtmlPageFactory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

FitNesse Default Page



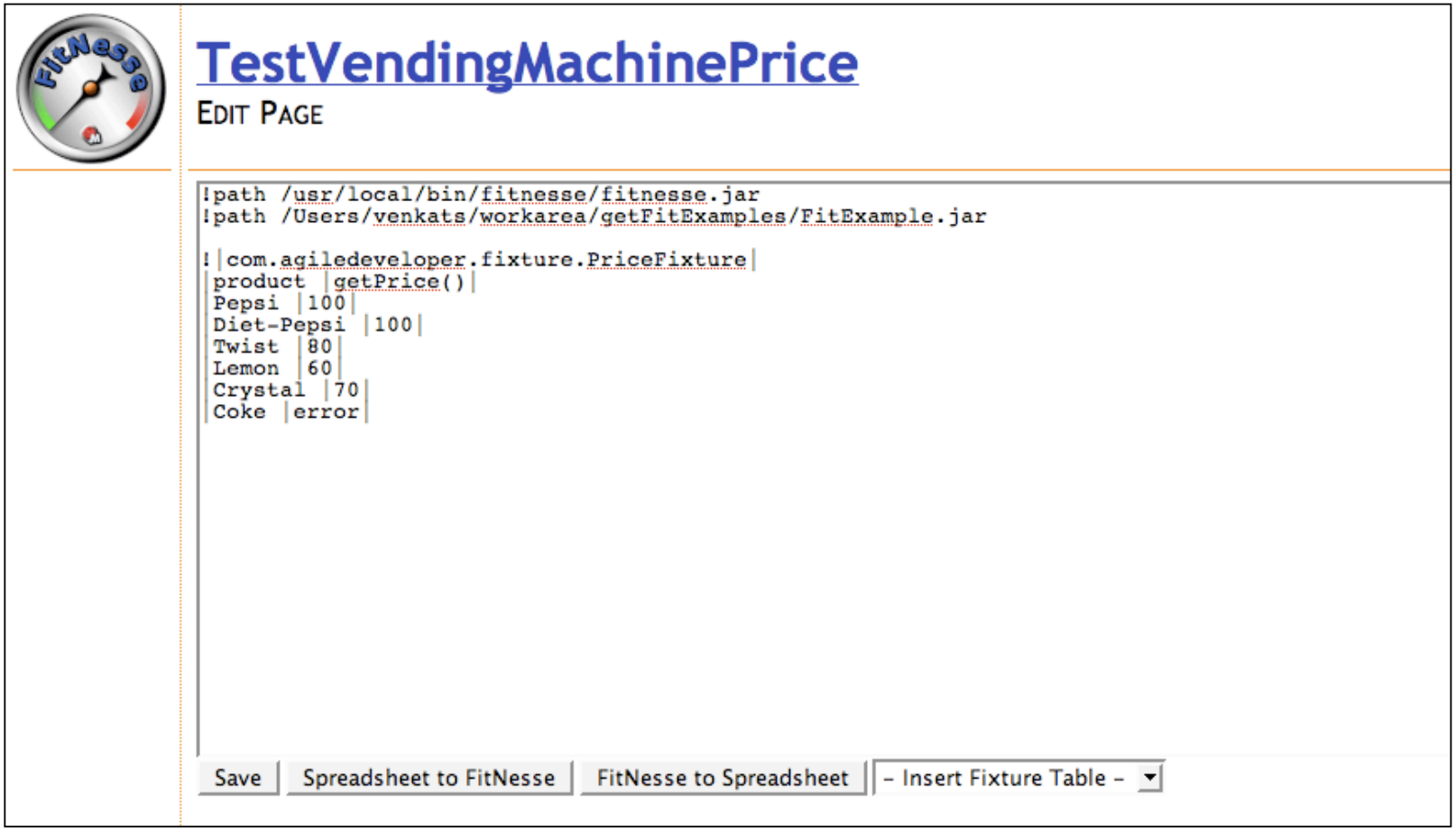
Click to Edit Page to add Test

Add A Test Page

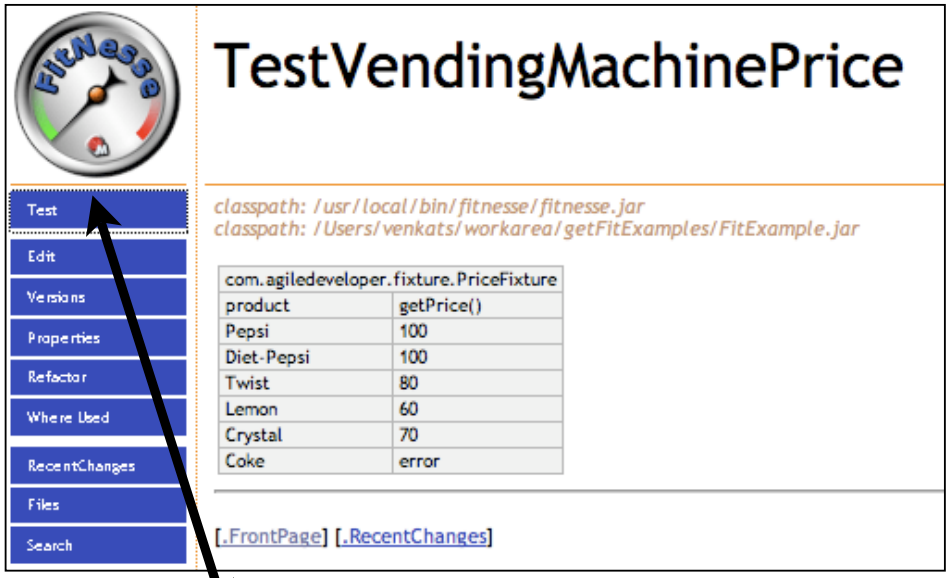


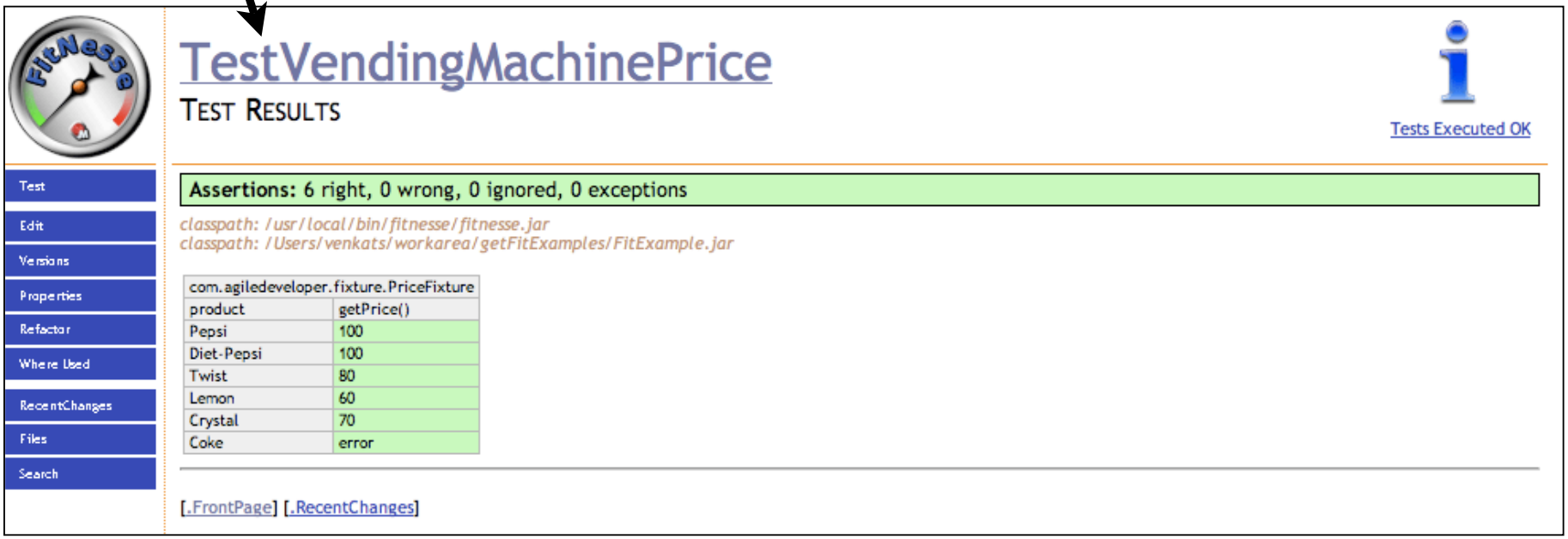
Click to Edit Page

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 -

Run Test


TestVendingMachinePrice
Test
Edit
Versions
Properties
Refactor
Where Used
RecentChanges
Files
Search
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
Test
Edit
Versions
Properties
Refactor
Where Used
RecentChanges
Files
Search
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]

Agenda

- Agile Development
- Testing
- Requirements
- Fit
- FitNesse
- FitLibrary
- Guidelines
- Conclusion

FitLibrary

- Alternate Fixtures
- Helps work with family of fixtures
- DoFixture helps test actions
- SetUpFixture helps gathering data at start of a test
- CalculateFixture for testing calculations
- ArrayFixture for testing ordered list
- SubsetFixture for testing unordered list

Agenda

- Agile Development
- Testing
- Requirements
- Fit
- FitNesse
- FitLibrary
- Guidelines
- Conclusion

Effects of Tests

- Tests are to help foster communication
- To express requirements through examples
- Tests will change as your understanding changes
- Automated tests help us know when things fall apart
- They make it possible to refactor and evolve the system

Effective Tests

- Keep it focused—weed out extraneous details
- Avoid duplication—Keep it DRY
- Make tests cohesive—focused on one business rule
- Keep tests isolated, don't expect any ordering
- Tests should not be brittle
- Tests must not be slow
- Automate your tests

Effective Tests...

- Give descriptive names for fixtures, tables, and columns
- Find meaningful examples
- Document as necessary
- Make it easier (and obvious) to understand
- Find easier ways (for understanding)
- Keep the tables relatively small with relatively small number of columns

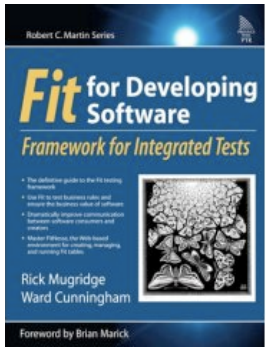
Agenda

- Agile Development
- Testing
- Requirements
- Fit
- FitNesse
- FitLibrary
- Guidelines
- Conclusion

Quiz Time

References

- http://fit.c2.com
- http://fitlibrary.sourceforge.net
- http://www.fitnesse.org



Thank You!

http://www.agiledeveloper.com — download