

Automatic Acceptance Testing with FIT / FitNesse

Filippo Ricca

Unità CINI at DISI

(Laboratorio Iniziativa Software FINMECCANICA/ELSAG spa -
CINI)

Genova, Italy

filippo.ricca@disi.unige.it

Marco Torchiano

Politecnico di Torino

Torino, Italy

marco.torchiano@polito.it

14/03/07

Automatic Acceptance Testir

1

Acceptance Testing

- ♦ **Acceptance Tests** are specified by the customer and analyst to test that the overall system is functioning as required (*Do developers build the right system?*).
- ♦ **Acceptance tests** typically test the entire system, or some large chunk of it.
- ♦ When all the **acceptance tests** pass for a given user story (or use case, or textual requirement), that story is considered complete.
- ♦ At the very least, an **acceptance test** could consist of a script of user interface actions and expected results that a human can run.
- ♦ *Ideally **acceptance tests** should be automated, either using the unit testing framework, or a separate acceptance testing framework.*

14/03/07

Automatic Acceptance Testing With FIT

2

Acceptance tests

- ♦ Used to judge if the product is acceptable to the customer
- ♦ Coarse grained tests of business operations
- ♦ Scenario/Story-based (contain expectations)
- ♦ Simple:
 - Happy paths (confirmatory)
 - Sad paths
 - Alternative paths (deviance)

14/03/07

Automatic Acceptance Testing With FIT

3

Unit Testing

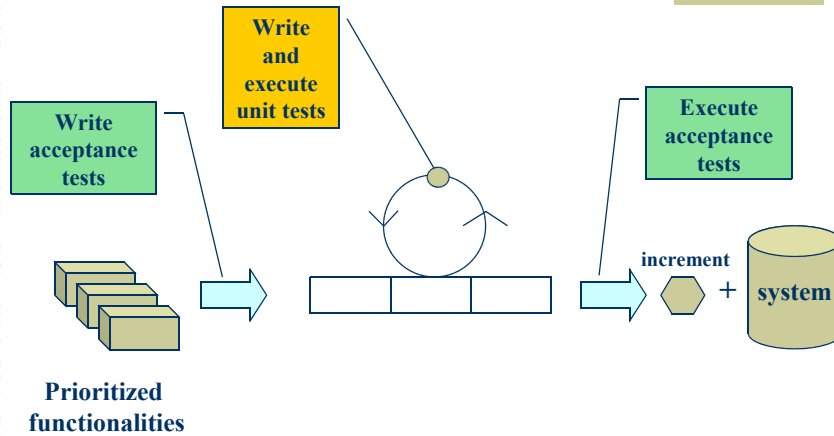
- ♦ **Unit Tests** are tests written by the developers to test functionality as they write it.
- ♦ Each **unit test** typically tests only a single class, or a small cluster of classes.
- ♦ ***Unit tests** are typically written using a unit testing framework, such as **JUnit** (automatic unit tests).*
- ♦ Target errors not found by Unit testing:
 - Requirements are mis-interpreted by developer.
 - Modules don't integrate with each other

14/03/07

Automatic Acceptance Testing With FIT

4

Iterative Software development



14/03/07

Automatic Acceptance Testing With FIT

5

Acceptance vs. Unit Testing

In theory:

Acceptance Tests	Unit Tests
Written by Customer and Analyst.	Written by developers.
Written using an acceptance testing framework (also unit testing framework).	Written using a unit testing framework.
(extreme programming) When acceptance tests pass, stop coding. The job is done.	(extreme programming) When unit tests pass, write another test that fails.
The motivation of acceptance testing is demonstrating working functionalities.	The motivation of unit testing is finding faults.
Used to verify that the implementation is complete and correct. Used for Integration, System, and regression testing. Used to indicate the progress in the development phase. (Usually as %). Used as a contract. Used for documentation (high level)	Used to find faults in individual modules or units (individual programs, functions, procedures, web pages, menus, classes, ...) of source code. Used for documentation (low level)
Written before the development and executed after.	Written and executed during the development.
Starting point: User stories, User needs, Use Cases, Textual Requirements, ...	Starting point: new capability (to add a new module/function or class/method).

6

Acceptance vs. Unit Testing

In reality: The difference is not so clear-cut.

- ♦ We can often use the same tools for either or both kinds of tests.

14/03/07

Automatic Acceptance Testing With FIT

7

Traditional approaches to acceptance testing

- ♦ **Manual Acceptance testing.** User exercises the system manually using his creativity.
- ♦ **Acceptance testing with “GUI Test Drivers”** (at the GUI level). These tools help the developer do functional/acceptance testing through a user interface such as a native GUI or web interface. “Capture and Replay” Tools capture events (e.g. mouse, keyboard) in modifiable script.

Disadvantages:
expensive, error prone, not repeatable, ...

Disadvantages:
Tests are brittle, i.e., have to be re-captured if the GUI changes.

“Avoid acceptance testing only in final stage: Too late to find bugs”

14/03/07

Automatic Acceptance Testing With FIT

8

Table-based approach for acceptance Testing

- Starting from a user story (or use case or textual requirement), the customer enters in a table (spreadsheet application, html, Word, ...) the expectations of the program's behavior.
- At this point tables can be used as oracle. The customer can **manually** insert inputs in the System and compare outputs with expected results.

	B	C	D	E	F	G
4	Team Name	Played	Won	Drawn	Lost	Rating
5	Arsenal	38	31	2	5	83
6	Aston Villa	38	20	2	16	54
7	Chelsea	38	35	1	2	93

Pro: help to clarify requirements, used in System testing, ...

Cons: expensive, error prone, ...

What is Fit?

- The Framework for Integrated Test (**Fit**) is the most well-known implementation (*open source framework*) of the table-based acceptance testing approach.
- Fit lets customers and analysts write “executable” acceptance tests using simple HTML tables.
- Developers write “fixtures” to link the test cases with the actual system itself.
- Fit compares these test cases, written using HTML tables, with actual values, returned by the system, and highlights the results with colors and annotations.

Using Fit

Just two steps are required to automate user acceptance tests using Fit:

- Express a test case in the form of a **Fit table**.
- Write the glue code, called a **Fixture**, that bridges the test case and system under test.

That's it!

You are all set to execute the tests automatically for the rest of the application's lifetime.

To work with Fit

To work with Fit, you must know and understand three basic elements:

- Fit table
- Fixture
- Test runner

Fit table

- ◆ A Fit table is a way of expressing the business logic using a simple HTML table.
- ◆ Fit tables help developers better understand the requirements and are used as acceptance test cases.
- ◆ Customers and Analysts create Fit tables using a tool like Word, Excel, or even a text editor.

14/03/07

Automatic Acceptance Testing With FIT

13

Fixture

- ◆ A fixture is an interface between the test instrumentation (in our case, the Fit framework), test cases (Fit tables), and the system under test.
- ◆ Fixtures are procedures/functions/classes usually written by developers.
- ◆ In general, there is a one-to-one mapping between a Fit table and fixture.

14/03/07

Automatic Acceptance Testing With FIT

14

Test Runner

- ◆ The Test runner compares the customer-set expectations with the actual results and reports any errors by color-coding the table rows: red for failures and green for passed tests.

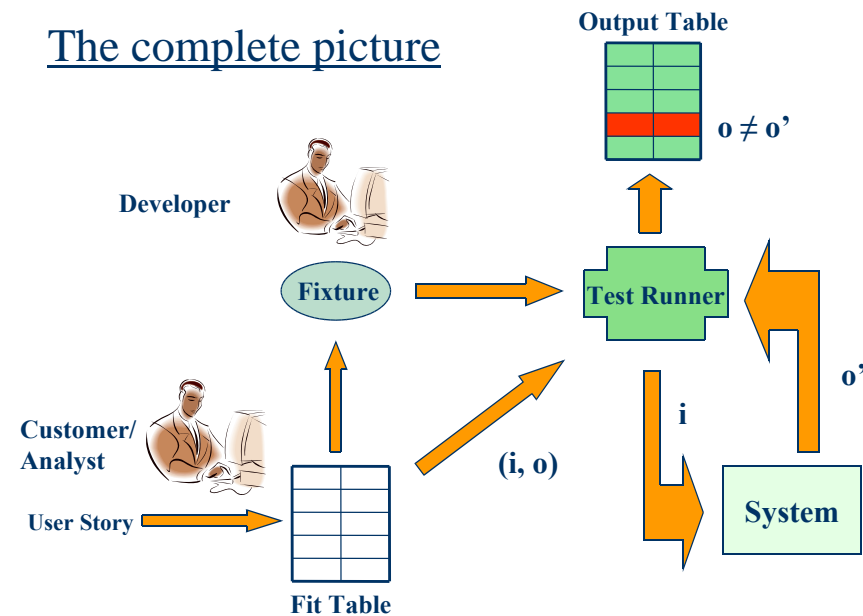
sample.VerifyRating					
team name	played	won	drawn	lost	rating()
Arsenal	38	31	2	5	83
Aston Villa	38	20	2	16	54
Chelsea	38	35	1	2	93
Dummy	38	35	1	2	100 <i>expected</i> 93 <i>actual</i>
Wigan	38	26	7	5	75

14/03/07

Automatic Acceptance Testing With FIT

15

The complete picture



Core Fixture

Fit provides three core fixtures:

- ◆ **Column fixture** for testing calculations.
- ◆ **Action fixture** for testing the user interfaces or workflow.
- ◆ **Row fixture** for validating a collection of domain objects. Used to check the result of a query.

Other fixtures are:

- ◆ **Summary fixture** to display a summary of all test on a page.
- ◆ **Html fixture** to examine and navigate HTML pages.
- ◆ **Table fixture, Command line fixture, ...**

Fit Tools

- ◆ **Fit** - <http://fit.c2.com/wiki.cgi?DownloadNow> (Java, C++, ...)
- ◆ **FitRunner** - <http://fitrunner.sourceforge.net/> (eclipse plug-in).
- ◆ **StoryTestRunner** - GUI for running Fit tests written with .NET
- ◆ **FitNesse** - <http://www.fitnesse.org> (Java, C#, P, C++).
- ◆ **“Other Fit tools for”**: .Net, Python, Perl, SmallTalk, C++, Ruby, Lisp, Delphi, etc.

What is a Wiki?

A minimalist Content Management System.

- ◆ Everyone can change every page.
- ◆ Changes are visible immediately.
- ◆ There are abbreviations for often used HTML tags.
- ◆ Whenever a word is combined of several others (TestDrivenDevelopment), it becomes a link to a new page. When the link is activated the first time, you can fill the empty page.

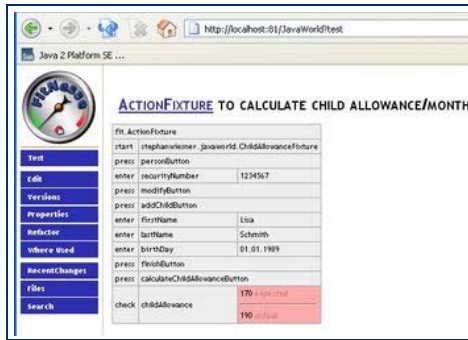
What is FitNesse?

A collaborative testing and documentation tool.

- ◆ It supports Java (*eclipse plug-in*), .Net, C++
- ◆ It Combines Fit with a Wiki Web for writing “natural language requirements” + Fit tables.
- ◆ It provides a simple way to run tests (Fit tables) and suits.
- ◆ It Supports **sub Wikis** for managing multiple projects.

How to use FitNesse?

- ◆ Install and start.
- ◆ Define the project on the FitNesse Wiki.
- ◆ Write requirements and fit tables on the FitNesse Wiki.
- ◆ Write the glue code (fixture), the unit tests and the business logic in your favorite IDE (eclipse).
- ◆ Execute the acceptance tests by a click on the Web page (*test button*).
- ◆ See the results (green or red) of executing the tests on the Web page.

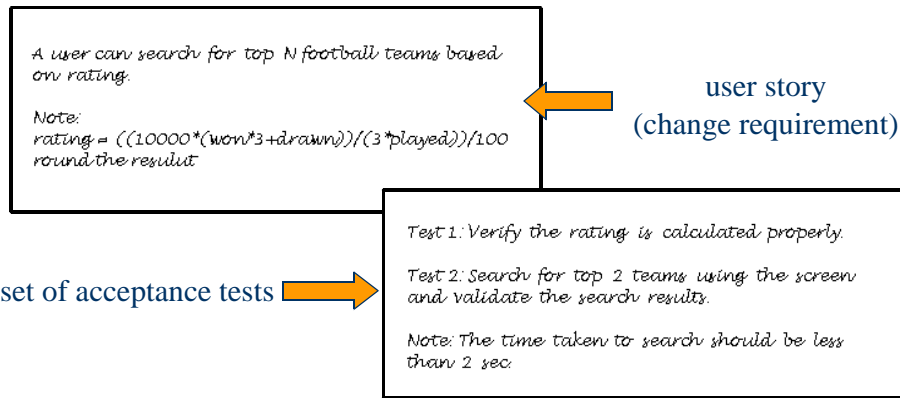


Sub Wikis and Test suites

- ◆ A normal Wiki is a collection of pages with a flat structure. All the pages are peers.
- ◆ FitNesse allows you to create Sub Wikis. Each Wiki page can be the parent of an entire new Wiki.
- ◆ A Test Suite executes all tests in the sub Wiki. SetUp and TearDown are invoked for each page of the suite.

An example: Football team Website

- ◆ A sports magazine decides to **add a new feature** to its Website that will allow users *to view top football teams based on their ratings*.
- ◆ An analyst and a developer get together to discuss the change requirements.
- ◆ The outcome of the discussion is:
 - a user story card that summarizes the change requirements
 - a set of acceptance tests
 - 3) an excel file with sample data



set of acceptance tests

user story
(change requirement)

Team Name	Played	Won	Drawn	Lost	Rating
Arsenal	38	31	2	5	83
Aston Villa	38	20	2	16	54
Chelsea	38	35	1	2	93

excel file with sample data

Test1: Fit Table

“verify the rating is calculated properly”

- For a team, given the number of matches played, won, drawn, and lost, we need to verify that the ratings are calculated properly.
- The first step is to express the logic using a Fit table. The table created using Excel during the requirements discussion can be easily converted into a Fit table by just adding a fixture name and modifying the labels.
- The **Fit table** on the right represents the first acceptance test case to verify the rating calculation.
- Now that we have created the Fit table, we need to write the glue code that will bridge the test case to the system under test.

“column fixture”

team name	played	won	drawn	lost	rating()
Arsenal	38	31	2	5	83
Aston Villa	38	20	2	16	54
Chelsea	38	35	1	2	93
Dumrry	38	35	1	2	100
Wigan	38	26	7	5	75

14/03/07

Automatic Acceptance Testing With FIT

25

Test1: Fixture

“verify the rating is calculated properly”

- For each input attribute represented by Columns 1 through 5 in the second row of the Fit table, there is a public member with the same name
- A public method public long rating() corresponds to the calculation in the sixth column.
- The rating() method in VerifyRating creates a Team object using the input data specified by the test case and returns the rating from the Team object; this is where the bridging between the test case and the system under test happens.

```
public class VerifyRating
    extends ColumnFixture {

    public String teamName;
    public int played;
    public int won;
    public int drawn;
    public int lost;

    public long rating() {
        Team team = new Team(teamName,
            played,won,drawn,lost);
        return team.rating;
    }
}
```

14/03/07

Automatic Acceptance Testing With FIT

26

The domain object representing a football team

```
package sport.businessObjects;
public class Team {
    public String name;
    public int played;
    public int won;
    public int drawn;
    public int lost;
    public int rating;

    public Team(String name, int ply, int won, int drawn, int lst) {
        super();
        this.name = name;
        this.played = played;
        this.won = won;
        this.drawn = drawn;
        this.lost = lost;
        calculateRating();
    }

    private void calculateRating() {
        float value = ((10000f*(won*3+drawn))/(3*played))/100;
        rating = Math.round(value);
    }
}
```

Test1: Running

“verify the rating is calculated properly”

- Here is what happens when you run the test:
- 3. Fit parses the table and creates an instance of sample.VerifyRating.
- 4. For each row in the table Fit set the values specified in Columns 1 through 5 to the corresponding fields in the fixture.
- 5. The rating() method is executed to get the actual value to be compared against the expected value specified in the sixth column.
- 6. If the expected value matches the actual value, then the test passes; otherwise it fails.

team name	played	won	drawn	lost	rating()
Arsenal	38	31	2	5	83
Aston Villa	38	20	2	16	54
Chelsea	38	35	1	2	93
Dumrry	38	35	1	2	100 expected ----- 93 actual
Wigan	38	26	7	5	75

passed
 failed
 exception

14/03/07

Automatic Acceptance Testing With FIT

28

Test 2

“Search for top two teams using the screen and validate the search results”

- Let's move on to the next test case: *Search for top two teams using the screen and validate the search results.*
- This step involves a screen (Web page) through which the user:
 - provides input
 - clicks on a button
 - seeks verification that the results returned match a collection of objects as expected.



14/03/07

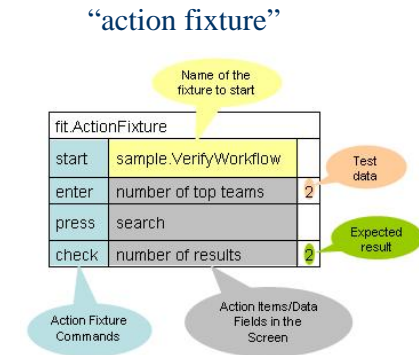
Automatic Acceptance Testing With FIT

29

Test 2: Fit table “1”

“Search for top two teams using the screen and validate the search results”

- Testing a screen using an **action fixture**: the action fixture supports four commands that you can use to simulate the actions that a user would perform on a screen. The commands are **start**, **enter**, **press**, and **check**.
- A typical usage of the screen could be described something like: *The user types 2 in the number of top teams text box, clicks on the Search button, and expects to see the top two teams displayed.* The following table represents this typical usage scenario in a way Fit can understand.



14/03/07

Automatic Acceptance Testing With FIT

30

Test 2: Fixture “1”

“Search for top two teams using the screen and validate the search results”

```
public class VerifyWorkflow extends Fixture {
    private int topN;
    private Collection<Team> results;

    public void numberOfTopTeams(int n){ topN = n; }

    public void search() {
        results = Populate.teams.getTopTeams(topN);
    }

    public int numberOfResults(){ return results.size(); }
}
```

14/03/07

Automatic Acceptance Testing With FIT

31

Test 2: Running “1”

“Search for top two teams using the screen and validate the search results”

- When you execute the test, here is what happens:
- Following the start command, the ActionFixture creates an instance of VerifyWorkflow and calls the **numberOfTopTeams (int n)** method with a parameter value of 2 to fulfill the enter command.
- Then it calls **search()** to fulfill the press command;
- then calls **numberOfResults()** to match the returned value against the value specified in the Fit table to complete the check command.

fit.ActionFixture		
start	sport.fixtures.VerifyWorkflow	
enter	number of top teams	2
press	search	
check	number of results	2

14/03/07

Automatic Acceptance Testing With FIT

32

Test 2: Fit table “2”

“validate the search results”

- ◆ The previous fixture doesn’t validate “completely” the search results. We have tested that the number of Teams is two; not that the two teams are “Chelsea” and “Arsenal”.
- ◆ Let’s express the “validate the search results” test case as a Fit table.

Name of the row fixture “row fixture”

sport.fixtures.VerifyResults					
name	played	won	drawn	lost	rating
Chelsea	38	35	1	2	93
Arsenal	38	31	2	5	83

Expected collection of objects

14/03/07

Automatic Acceptance Testing With FIT

33

Test 2: Fixture “2”

“validate the search results”

```
public class VerifyResults extends RowFixture {  
    public Object[] query() throws Exception {  
        return Populate.teams.getTopTeams(2).toArray();  
    }  
  
    public Class getTargetClass() {  
        return Team.class;  
    }  
}
```

14/03/07

Automatic Acceptance Testing With FIT

34

Test 2: Running “2”

“validate the search results”

- ◆ The VerifyResults class extends RowFixture and overrides the query() and getTargetClass() methods.
- ◆ The query() method interacts with SearchUtil (the system under test) and returns an array of objects that represents the actual results.
- ◆ The getTargetClass() method returns Team.class, the type of the domain object represented by the test case under consideration.
- ◆ Fit uses the object array returned by the query method to match against the expected results specified in the Fit table.
- ◆ The test results produced by Fit are shown on the right.

sample.VerifyResults					
name	played	won	drawn	lost	rating
Chelsea	38	35	1	2	93
Arsenal	38	31	2	5	83

14/03/07

Automatic Acceptance Testing With FIT

35

Testing non-functional requirements using “timed action fixture”

- ◆ What about the time required for completing the search? How do we test the response time?
- ◆ Just use fit.TimedActionFixture, a descendant of fit.ActionFixture.
- ◆ **TimedActionFixture provides visual feedback on how long the functions take to execute.** The result produced by TimedActionFixture includes the time when each command begins and how long it takes to execute, referred to as split.

14/03/07

Automatic Acceptance Testing With FIT

36

Summary fixture

- ◆ You can use `fit.SummaryFixture` to provide a summary of the test results on the output.
- ◆ The Fit table for summary fixture is a one-by-one table containing the fixture name.
- ◆ Usually it is included as the last Fit table on the input file so that it can produce a collective summary of all the tests on a given input file.

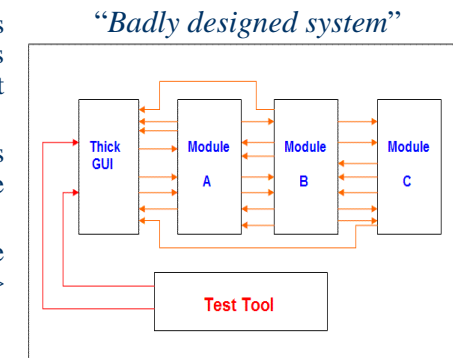
14/03/07

Automatic Acceptance Testing With FIT

37

Badly designed systems makes testing difficult

- ◆ We have a thick GUI that has program logic. The interfaces between the modules are not clearly defined.
- ◆ Testing of specific functions (Unit Testing) cannot be isolated.
- ◆ Testing has to be done through the GUI => Fit/Fitnesse is not sufficient.
- ◆ Testing is difficult.



Test Tool = “GUI Test Drivers” or Fit/Fitnesse + Abbot

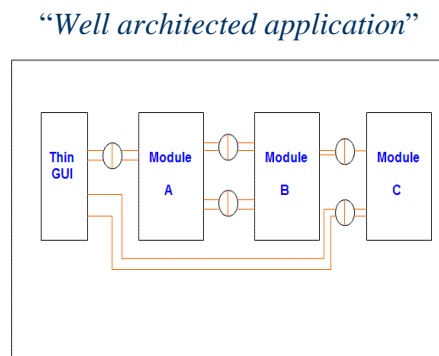
14/03/07

Automatic Acceptance Testing With FIT

38

Well architected applications makes testing simple

- ◆ The GUI does not contain any program logic other than dealing with presentation.
- ◆ The interfaces between the modules are well defined.
- ◆ This give us testing advantages. Unit and System acceptance testing are simpler.



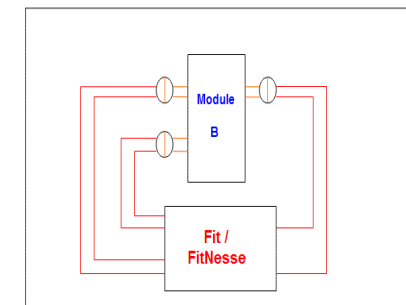
14/03/07

Automatic Acceptance Testing With FIT

39

Testing a module in a well designed application

- ◆ When an application has modules with well defined interfaces, each module can be tested independently from the other modules.
- ◆ Using this type of environment the developer can test the module to make sure everything is working before trying to integrate it with other modules.
- ◆ This system does not require Fit/FitNesse. You could use any automated test harness that works for your application (i.e., Junit).



Test Tool = Fit/Fitnesse or Junit

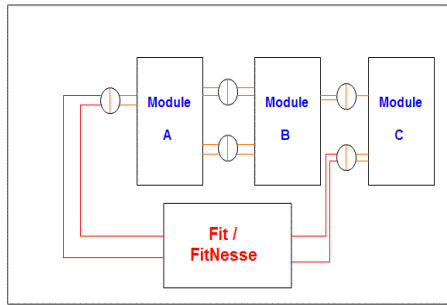
14/03/07

Automatic Acceptance Testing With FIT

40

System/Integration/Acceptance testing of a Well architected application using Fit/Fitnesse

- ◆ You could put data into Module A using (for example) an action fixture and collect the output of Module C.

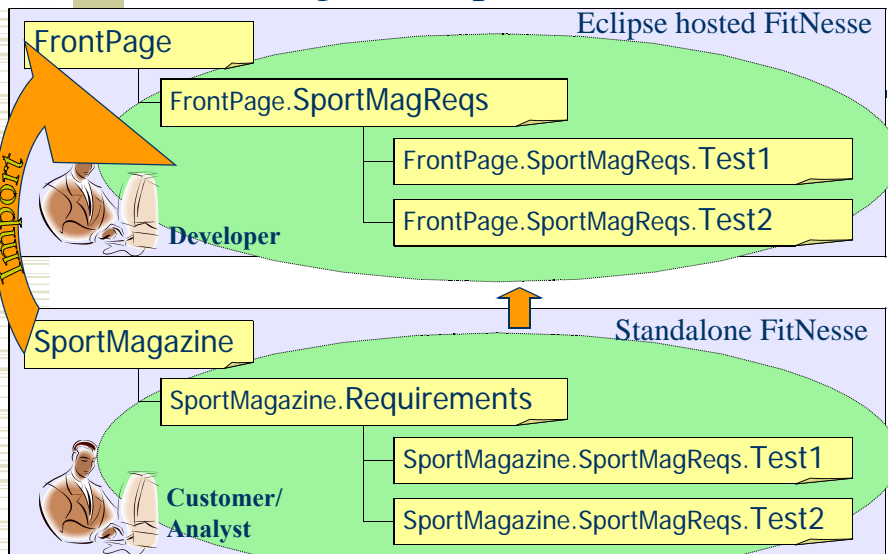


Test Tool = Fit/Fitnesse

Team work organization

- ◆ Customer, analysts and project manager work with the main FitNesse server
 - Writing and updating requirements and tests
 - Monitoring the progress
- ◆ Developers work within their development environment
 - Importing pages from the main wiki
 - Developing and running tests locally
 - Eventually committing new versions

Structuring work spaces



Main FitNesse (standalone)

SportMagazine Sub-wiki

```

!1 Requirements for rated team search

!path fitness.jar
!path FitNesseRoot\files\sport.jar

You should download the latest
http://files/sport.jar

To upload a newer version (e.g. including
updates) you should go http://files/

!contents
    
```

Conclusions

- ♦ Badly designed systems makes testing difficult. Unit testing is complex and all end-to-end tests are through the GUI.
- ♦ Well architected applications simplify testing. Unit testing is simple and end-to-end tests are through interfaces of modules.
- ♦ Manual acceptance testing is expensive, error prone and not repeatable.
- ♦ Fit/Fitnesse is a tool to write, organize and execute table-based tests.
- ♦ Fit tables help to clarify “textual requirements”.
- ♦ Fit tables “are requirements verifiable and executable”.
- ♦ The motivation of Fit/Fitnesse testing is demonstrating working functionalities.
- ♦ The motivation of Junit is finding faults.
- ♦ Fit is compatible with Junit.
- ♦ Fit/Fitnesse can be useful for Managers, Customers, Analysts and Developers.

Fit helps to clarify requirements

- ♦ It is estimated that 85% of the defects in developed software originate in the requirements (communication between customer and analyst, communication between analyst and developer).
- ♦ There are several “sins” to avoid when specifying requirements: noise, silence, ambiguity, over-specification, wishful thinking, ... => ambiguous, inconsistent, unusable requirements.

“order-processing system for a brewery”

if a retail store buys 50 cases of a seasonal brew, no discount is applied; but if the 50 cases are not seasonal a 12% discount is applied. If a store buys 100 cases of a seasonal brew, a discount is applied, but it's only 5%. A 100-case order of a non-seasonal drink is discounted at 17%. There are similar rules for buying in quantities of 200.

list price per case	number of cases	is seasonal	discount price()	discount amount()
10	10	true	100	0
10	10	false	95	5
10	50	true	500	0
10	50	false	440	60
10	100	true	950	50
10	100	false	830	170
10	200	true	1800	200
10	200	false	1600	400

Manager, Customer, Analyst, developer perspectives

Manager Perspective	Customer Perspective	Analyst Perspective	Developer Perspective
Used to verify that the implementation is complete and correct.	Used to understand better the requirements.	Used to specify (clarify) requirements	Used to understand better the requirements
Used to indicate the progress in the development phase. (Usually as %).	Used to indicate the progress in the development phase. (Usually as %).	Used to understand better the requests (needs) of the customer.	Used for TDD (Test Driven Development).
Used as a contract.	Used as a contract.		Used for System testing and regression testing
			Used as documentation (maintenance intervention)

References

- **Demo:** <http://softeng.polito.it/courses/tutorial/FitnesseInEclipse.html>
- **Eclipse FitNesse plug-in:** <http://www.bandxi.com/fitnesse/download.html>
- **Acceptance testing.** J. Aarniala, University of Helsinki. www.cs.helsinki.fi/u/jaarnial/jaarnial-testing.pdf, Helsinki, October 30, 2006.
- **Fit for analysts and developers, Test-first development from the user perspective.** N. Jayaratchagan, JavaWorld.com, <http://www.javaworld.com/javaworld/jw-06-2006/jw-0612-fit.html>, December 6, 2006.
- **Fit/FitNesse Implementation Strategy.** J. Matthews, <http://awta.wikispaces.com/page/diff/Fit+-+Fitness+Implementation+Strategy>, Jan 23, 2007.
- **XP Testing a GUI with FIT, Fitnesse, and Abbot.** Joseph Bergin, Pace University, <http://www.csis.pace.edu/~bergin/xp/guitesting.html>.
- **<http://fit.c2.com/>.** WebSite devoted to the distribution and further development of a testing framework by Ward Cunningham.
- **<http://www.fitnesse.org>.** Wiki-based Integrated Development Environment for Fit.
- **Test-driven development.** C. Steindl. <http://www.agilealliance.org/system/article/file/1423/file.pdf>