Acceptance testing

An introduction

Alessandro Marchetto

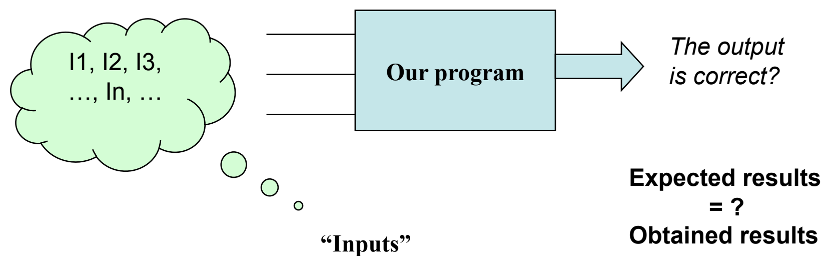
Fondazione Bruno Kessler - IRST

Outline

- Introduction
- Acceptance and Unit testing
- Table-based testing and Fit/Fitnesse
- Fit/Fitnesse: An Example

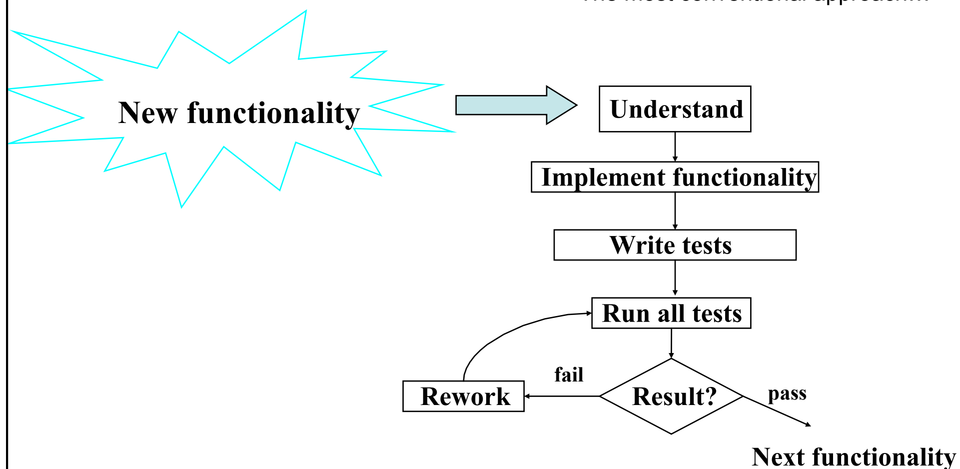
Testing

- One of the practical methods commonly used **to detect the presence of errors (*failures*)** in a computer program is to test it for a set of inputs.



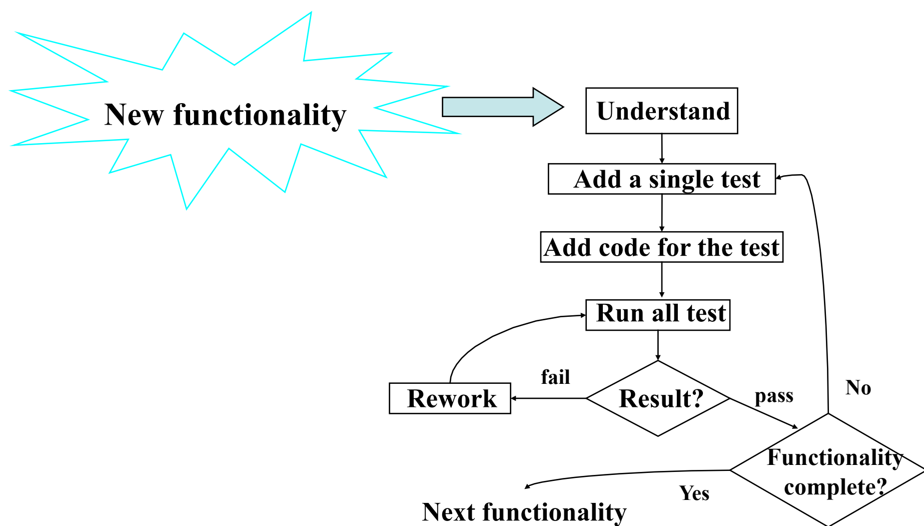
Test last

The most conventional approach…

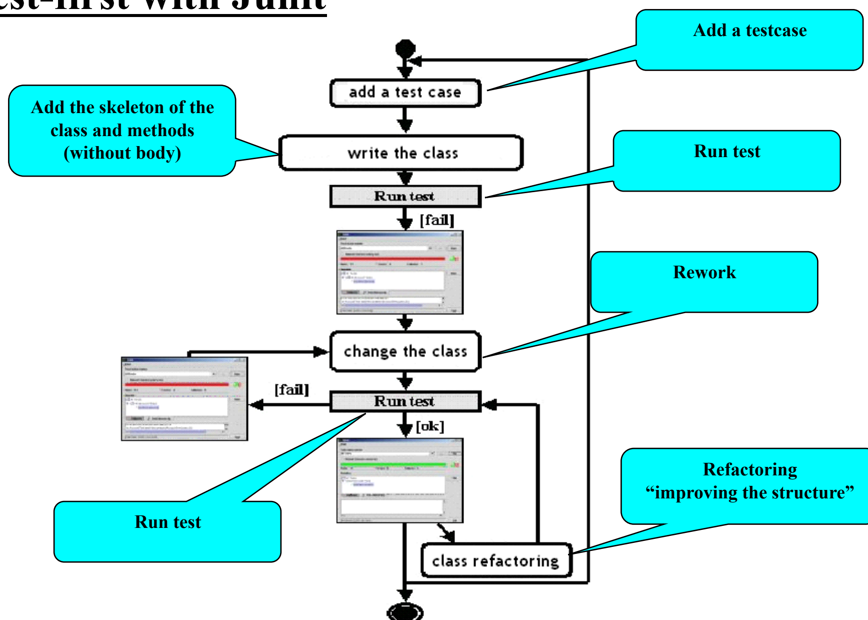


Test first

“Extreme programming” (**XP**) champions the use of tests as a development tool …



Test-first with Junit

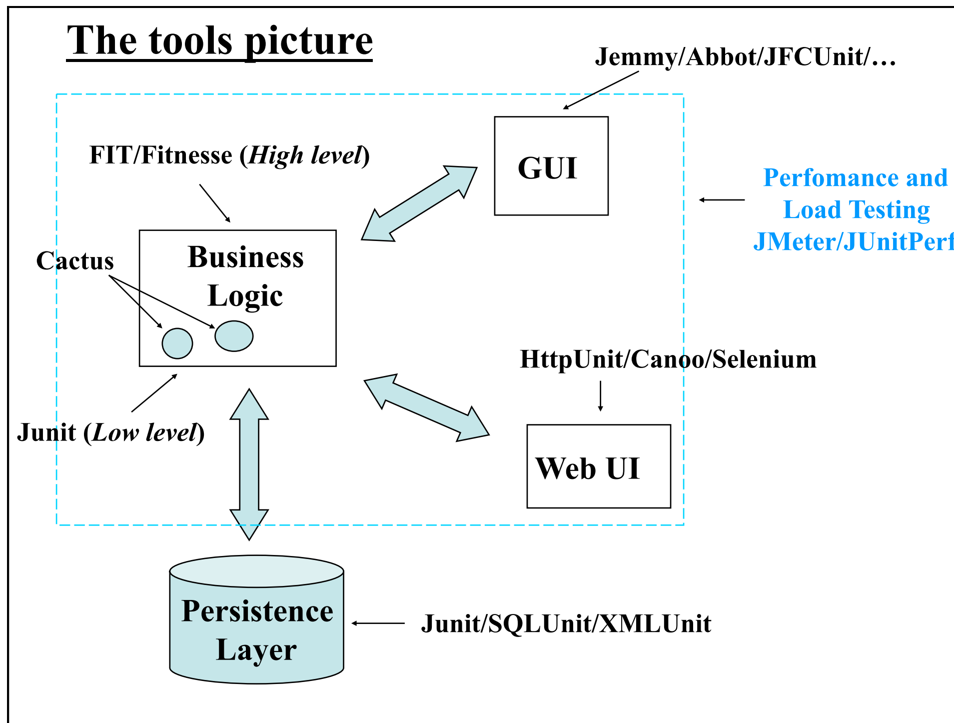


Test first advantages

- Each method has associated a test-case
  - the confidence of our code increases …
- It simplifies:
  - refactoring
  - restructuring
  - maintenance
  - the introduction of new functionalities
- Test first helps in writing the documentation
  - test cases are good “use samples”

Test levels

- **Unit testing** – this is basically testing of a single function, procedure, class.
- **Integration testing** – this checks that units tested in isolation work properly when put togheter.
- **System testing** – here the emphasis is to ensure that the whole system can cope with **real data**, monitor **system performance**, test the system’s error handling and recovery routines.
- **Acceptance testing** – this check that the overall system is functioning as required



Outline

- Introduction
- Acceptance and Unit testing
- Table-based testing and Fit/Fitnesse
- Fit/Fitnesse: An Example

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.
- 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.*

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 do not integrate with each other

Acceptance vs. Unit Testing

In summary:

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, … (the whole system)	Starting point: new capability (to add a new module/function or class/method). …. (the unit)

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 to 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, …

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

Outline

- Introduction
- Acceptance and Unit testing
- Table-based testing and Fit/Fitnesse
- Fit/Fitnesse: An Example

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

inputs

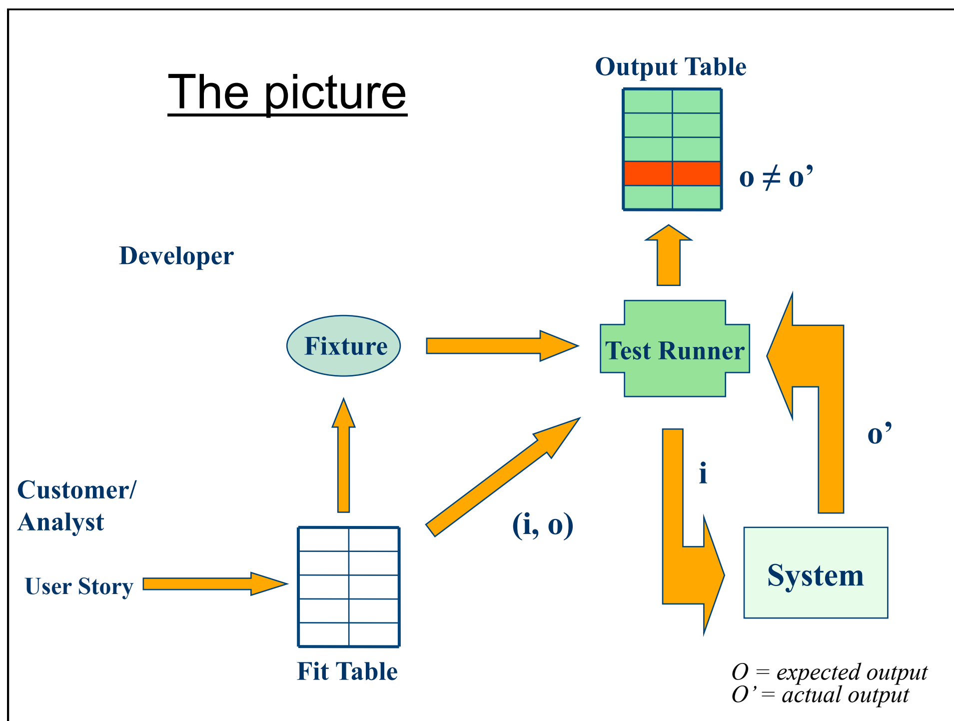
output

Pro: help to clarify requirements, used in System testing, …
Cons: expensive, error prone, …

What is Fit?

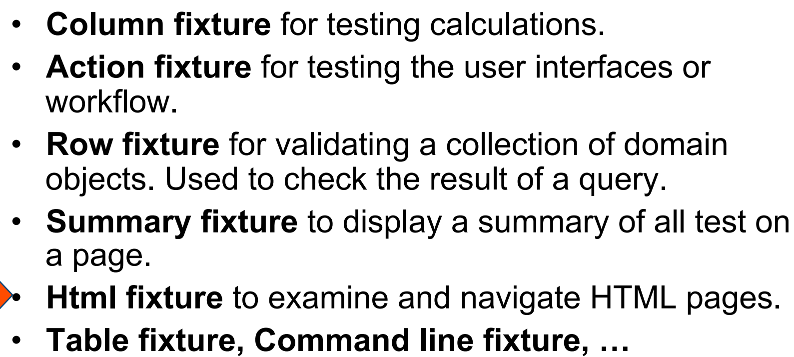
- The Framework for Integrated Test (**Fit)** is the most well-known implementation (*open source framework*) of the table-based acceptance testing approach.
- Fit lets customers and analysts write "executable" acceptance tests by means of tables written using simple HTML.
- Developers write "**fixtures**" 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.

The picture



Fixture

Fit provides a set of 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.
- **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, …**

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 **Wiki** and **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.

"expected 170, actual 190"

Outline

- Introduction
- Acceptance and Unit testing
- Table-based testing and Fit/Fitnesse
- Fit/Fitnesse: An Example

An example: the Football-team Application

- A sports magazine decides to **add a new feature** to its Website that will allow users *to view top football teams according to 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
  - an excel file with sample data

A user can search for top N football teams based on rating.

Note:
rating = ((10000*(won*3+drawn))/(3*played))/100
round the result

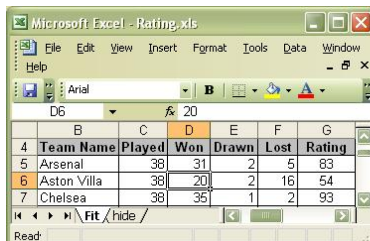
user story
(change requirement)

set of acceptance tests

Test 1: Verify the rating is calculated properly.

Test 2: Search for top 2 teams using the screen and validate the search results.

Note: The time taken to search should be less than 2 sec.



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

The domain object representing a football team

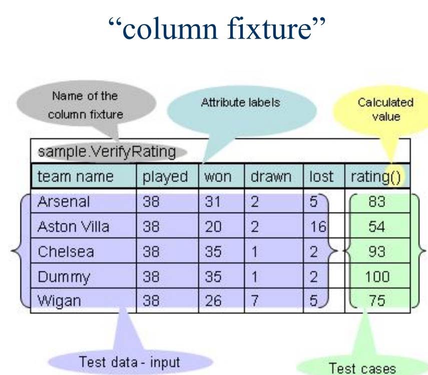
```
package sport.businessObjects;
public class Team {
   public String name;
   public int played;
   public int won;
   public int drawn;
   public int lost;
   public int rating;

   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: 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 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"



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
Wigan	38	26	7	5	75

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;
   }
}
```

Test1: Running
“verify the rating is calculated properly”

- Here is what happens when you run the test:

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

Launch the test runner …

sport.fixtures.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 expected ------------------- 93 *actual*
Wigan	38	26	7	5	75

passed

failed

exception