Catalysts

Test-Driven Development at the Acceptance Testing Level

Dr. Christoph Steindl
steindl@catalysts.cc

A feature is not specified …

Catalysts

… until its acceptance test is written

Who writes these acceptance tests?

- Business Analyst for the happy path ("clean tests")
- QA Test Writers and Testers for the error conditions and boundary cases ("dirty tests")

These acceptance tests are

- written by the stakeholders
- written in a very high level language
- automated
- should be executed frequently

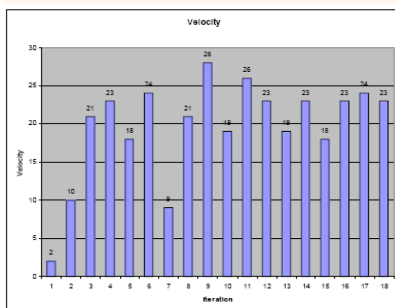
A feature is not done …

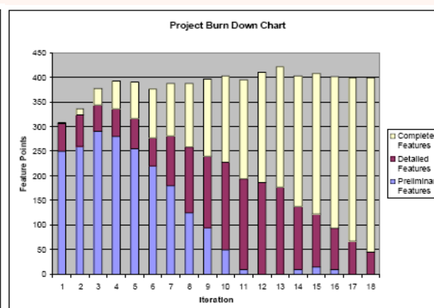
Catalysts

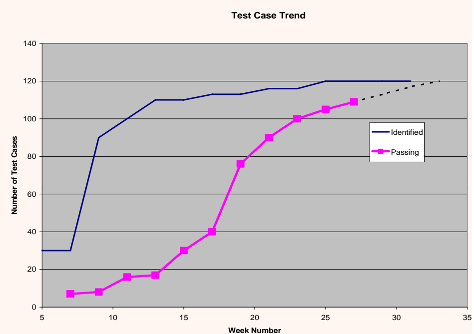
… until all its acceptance tests pass

Knowing when a feature is done allows us to visualize the

- velocity (how many features are implemented each iteration)
- features (how many features are completed, detailed, preliminary)
- test cases (how many are identified, passing)







Manual Approach to Acceptance Testing

Catalysts

The user exercises the system manually
using his creativity.

But:

- The developers don‘t know the goal, the tests that the system has to pass
- This approach does not support Test-First-Design
- This approach is expensive (due to manual effort which has to be repeated whenever the system changes)
- Errors may be overlooked (no automated verification whether the actual matches the expected)
- In a crunch, this kind of testing is easily cut back
- There can be big arguments about the pass/fail decision

GUI Capture & Replay Approach to Acceptance Testing

Catalysts

Capture user events (mouse, keyboard) in a modifiable script and replay them later.

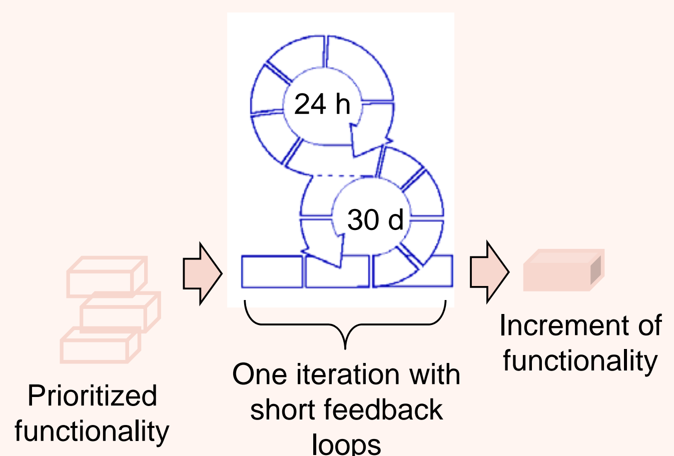
But:

- The GUI has to exist, so this approach does not support Test-First-Design
- The tools are expensive
- The tests are brittle, have to be re-captured if the system changes (the tests are less brittle if it is possible to abstract screen coordinates to GUI objects)
- The GUI may become cast in stone. The GUI may not be updated any longer because the tests would fail.

Iterative Software Development

Catalysts

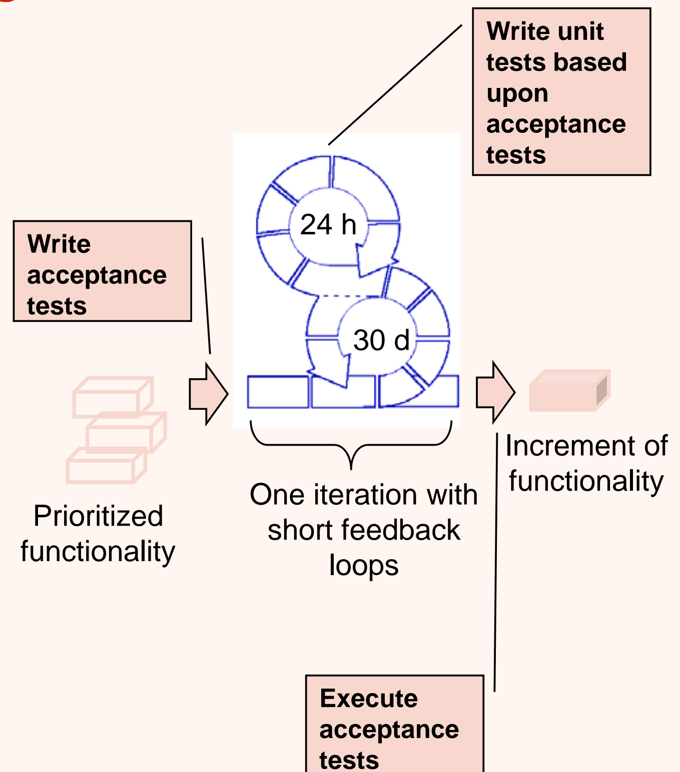
- At the start of the iteration,
  - the customer explains the expected functionality to the team,
  - the customer sets the priorities,
  - the team estimates the effort.
- Then the team
  - thinks about the necessary tasks to implement the functionality,
  - details the estimation, and
  - self-selects the tasks.
- During the iteration, the team holds 15 minutes status meetings in order to discuss the current tasks, the achievements and the problems.
- At the end of the iteration, the team demonstrates to the customer the increment of potentially shippable functionality.



Agile Approach to Acceptance Testing

Catalysts

- Don‘t wait until the software has been written in order start with acceptance testing.
- Write the acceptance tests as soon as possible.
- Build the unit tests upon the acceptance tests.
- Execute the acceptance tests during the iteration to understand the progress.
- Execute the acceptance tests at the end of the iteration for verifying that the requested functionality has been built.



When best to write the Acceptance Tests?

Catalysts

- Talk to the stakeholders days before each iteration
- Write initial acceptance tests for the features they plan to schedule
- Elaborate those acceptance tests once scheduled

When best to run the Acceptance Tests?

Catalysts

- Continuously
- At every check-in
- So that you immediately recognize if a change breaks any existing tests
- With continuous integration

Who runs the Acceptance Tests?

Catalysts

- Developers
- Testers
- Managers
- Stakeholders

- The Continuous Build Environment
  - runs them automatically
  - creates charts automatically
  - displays the results visibly

Benefits of this approach

Catalysts

- Better collaboration between customer and developer, faster feedback
- Better involvement of the customer / business / user / domain expert
- Specification of the requirements in the language of the business
- Focus on the scenarios, the flow of events, the dynamic behavior
- Tests in an executable form (where the execution can be automated)
- Specification of the tests before the requirements are implemented (can be used to verify the requirements)

- Targets errors not found by unit testing
  - Requirements are mis-interpreted by developer
  - Interface of software is not as intended
  - Modules don't integrate with each other

FIT-Framework for Acceptance Tests

Catalysts

- FIT is an open source framework (under the GPL) created by Ward Cunningham (http://fit.c2.com)
- The customer write tests as HTML tables
  - for data-driven tests (input – processing – output)
  - for scenario tests
- The framework parses the HTML, runs the tests, captures the results and outputs them as a modified HTML document
- The customer documents the tests with free text between the tables (which is ignored by the framework)

- It is easy for the customer to describe the requirements (no programming, just text).
- Tests are written before the code, so this approach supports Test-First-Design
- The framework is for free; it supports Java, .NET, Python and other programming languages
- The execution of the tests can be automated.
- But: no capture & replay possible

How to Use FIT?

Catalysts

eg.Division		
numerator	denominator	quotient()
1000	10	100.0000
-1000	10	-100.0000
1000	7	142.85715
1000	.00001	100000000
4195835	3145729	1.3338196

- **Fixtures** are types of HTML tables with a specific behavior of interpreting the values in the table.
- **ColumnFixture**: maps columns in the test data to fields or methods; a new column fixture object is created for each table.
- **ActionFixture**: executes the command in the first column
  - **`start`** `aClass`: creates an object of the specified class
  - **`enter`** `aMethod anArgument`: invokes the specified method on the object (with the specified parameters)
  - **`press`** `aMethod`: invokes the specified method on the object (without parameters)
  - **`check`** `aFunction aValue`: invokes the specified function (without parameters) and compares the return value with the specified value

fit.ActionFixture		
start	eg.music.Browser	
enter	library	Source/eg/music/Music.txt
check	total songs	37

- **RowFixture**: checks the result of a query and compares the returned records with the records specified in the table (as rows)
  - **binds** the columns to variables and methods
  - **executes the query** to get the result records
  - **matches** the expected and result records
  - **marks** missing and surplus records (i.e. rows)
  - **highlights** incorrect output (i.e. cells)

eg.music.Display					
title	artist	album	year	time()	track()
Scarlet Woman	Weather Report	Mysterious Traveller	1974	5.72	6 of 7
American Tango	Weather Report	Mysterious Traveller	1974	3.70	2 of 7

What is a Wiki?

Catalysts

- A minimalistic Content Management System
  - Everyone can change every page
  - Changes are visible immediately (but are under version control in case that you damage something)
  - There are abbreviations for often used HTML tags
  - Whenever a word is combined of several others (TestFirstDesign), it becomes a link to a new page. When the link is activated the first time, you can fill the (originally) empty page.
- The first Wiki was invented by Ward Cunningham: http://c2.com/cgi/wiki

What is FitNesse?

Catalysts

- An open source framework (under the GPL) created by Robert C. Martin et al. (http://fitnesse.org)
  - Supports Java, .NET, C++
  - Combines FIT with a Wiki Web for writing the Fixtures (HTML tables)
  - Versions pages, searches pages, supports simple refactorings (rename, move, delete page)
- A collaborative testing and documentation tool
- It provides a *very simple* way for teams to collaboratively create documents, specify tests, and run those tests and suites of those tests.
- A web server:
  - It requires **no configuration or setup**.
  - Just run it and then direct your browser to the machine where it is running.
- A wiki, so you can *easily* create new pages, hyperlinks, lists, tables.

How to use FitNesse?

Catalysts

- Install and start
- Define project on the FitNesse Wiki
- Write acceptance tests on the FitNesse Wiki.
- Write the glue code, the unit tests and the business logic in your favorite IDE.
- Execute the acceptance tests.

- But:
  - Not so tightly integrated into the automated build process, i.e. no test coverage computed out of the box,…
  - Not so tightly integrated into the IDE, i.e. no end-to-end debugging

Standard FitNesse Fixtures   Catalysts

- ColumnFixture: operates on a single object; each row loads a data structure (domain object) and then invokes functions upon it, often used for test object creation.
- ActionFixture: to write a script that emulates a user interface
- RowFixture: to match all the rows from a simple query, independent of order. Each row is the data of a domain object, all rows are matched, missing and surplus rows are reported; often used to check the results of a query (where the query is built into the fixture or taken from a known static variable)
- SummaryFixture: displays a summary of all tests on a page; often added to TearDown
- RowEntryFixture: (like ColumnFixture) to add a bunch of data to a database, or to call a function over and over again with different arguments.
- TimedActionFixture: (like ActionFixture) additionally with visual feedback on how long certain functions take to execute
- ParametricRowFixture: (like RowFixture) additionally you can pass arguments into the RowFixture
- CommandLineFixture: to execute shell commands in multiple threads
- HtmlFixture: to examine and navigate HTML pages

Sub Wikis and Test Suites   Catalysts

- A normal wiki is a collection of pages with a flat structure. All the pages are peers. You add a top-level page simply by placing a WikiWord on an existing top-level page, and then clicking on the ?
- FitNesse allows you to create sub wikis. Each wiki page can be the parent of an entire new wiki. You create a sub wiki page by *ParentPage.SubPage* (or the shortcut ^*SubPage*), and then clicking on the ?
- Each wiki (and sub wiki) can have its own
  - ClassPath
  - PageHeader, PageFooter
  - SetUp, TearDown
  - SuiteSetup, SuiteTearDown
- Test Suites
  - A Test Suite executes all tests in the sub wiki (tree of pages)
  - SetUp and TearDown pages are invoked for each page of the suite
  - To wrap an entire suite, define the operations on pages SuiteSetUp and SuiteTearDown

Importing a (Sub) Wiki

Catalysts

- In a development environment, you typically have a main FitNesse server where all the tests are maintained and new tests are added.
- However, it is not practical for all the developers to run the tests on one server. Therefore each developer should have a local FitNesse installation in his / her development environment.
- The test pages from the main server get imported by each developer so they can execute the most recent tests against their current code base.

- You create a page or use an existing page that will contain the imported wiki by
  - opening the *properties* view of this page
  - supplying the URL to the remote wiki that you'd like to import
  - optionally selecting to automatically update the imported content when executing tests
- You can edit the page locally or remotely

References

Catalysts

- Robert C. Martin: *XP Immersion*, Workshop at the JAOO Conference, 2005.

- Rick Mugridge, Ward Cunningham: *Fit for Developing Software*, Prentice Hall PTR, 2005.
- Stefan Roock: *Akzeptanztests mit FIT und Fitnesse*, http://www.stefanroock.de/downloads/Fitnesse.pdf
- Frank Westphal: *Testgetriebene Entwicklung mit Junit und FIT*, dpunkt.verlag, 2005.

- Kent Beck: *Test-Driven Development: By Example*, Addison-Wesley, 2002.
- David Astels: *Test-Driven Development: A Practical Guide*, Prentice Hall, 2003.
- Vincent Massol: *Junit in Action*, Manning Publications, 2003.
- J. B. Rainsberger: *Junit Recipes*, Manning Publications, 2004.
- Andrew Hunt, David Thomas: *Pragmatic Unit Testing*, Pragmatic Bookshelf, 2004.
- Johannes Link, Peter Fröhlich: *Unit Tests mit Java*, dpunkt.verlag, 2002.

Online References

Catalysts

- FIT: http://fit.c2.com/
  - http://fit.c2.com/wiki.cgi?JavaDownloads
  - http://fit.c2.com/wiki.cgi?DotNetDownloads
- Fitnesse: http://fitnesse.org/