

Automated Acceptance Testing

Björn Beskow

Callista Enterprise AB

bjorn.beskow@callista.se

<http://www.callista.se/enterprise>





Target audience and Objectives

□ Target audience

- Developers, Designers, Architects, Project Managers and Project Sponsors interested in lean and mean ways to achieve good-enough quality without paying an excessive price

□ Objectives

- Provide an overview of some Acceptance Testing strategies, tools and frameworks

□ Non-Objectives

- To cover Unit Testing, Performance Testing, Stress Testing, ...



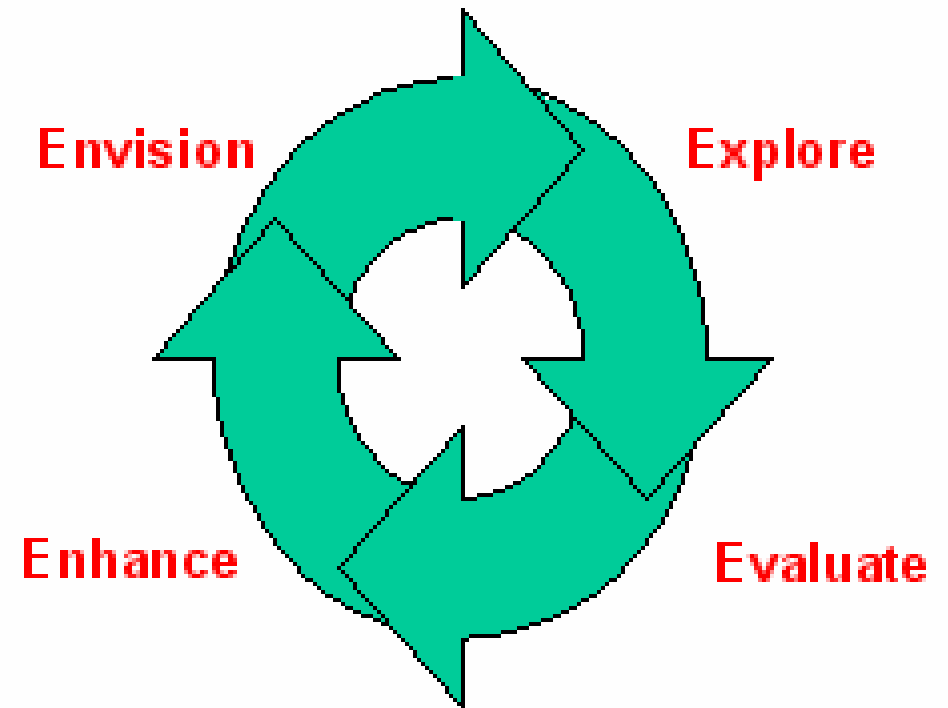
Agenda

- Background: The case for automated tests
- Classification of automated tests
- Tools and Frameworks for Acceptance Tests
- Data-driven Tests



Testing - a critical success factor for agility

- ❑ Enables true iterativity
- ❑ Enables relentless refactoring
- ❑ Shows expectations and elicits feedback
- ❑ Demonstrates visible progress





Quality Assurance precedes Quality Assessment

- ❑ Testing is about Quality Assurance, not just Quality Assessment
- ❑ Quality Assessment only indirectly affect quality
- ❑ Testing *reveals information*
- ❑ Testing helps *focus project activity*





Test Automation Goals

Tests should be S.M.A.R.T:

- Self Checking
- Maintainable
- Act as documentation
- Repeatable and Robust
- To the point – provide “defect triangulation”



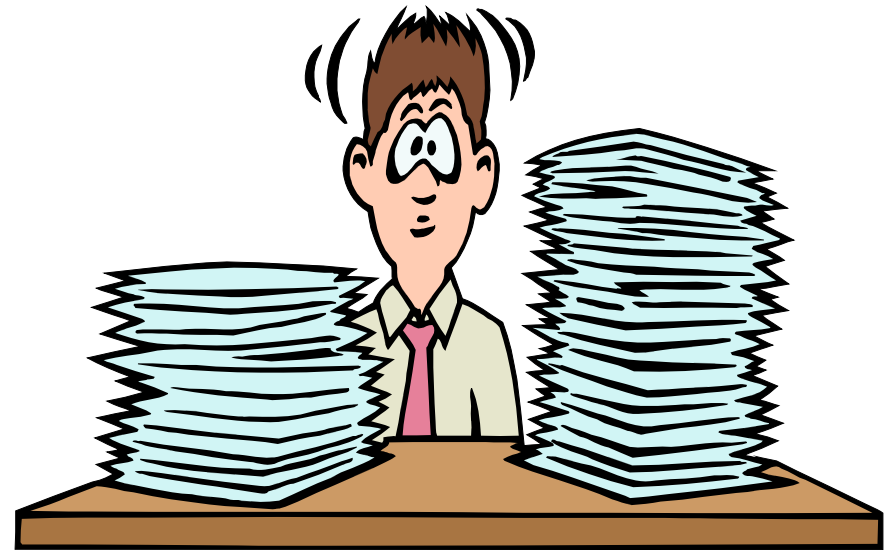


Manual Tests are ...

- Repetitive
- Error-prone
- Difficult to test other units than the User Interface

yet ...

- a Manual Test Process must be present in order to automate it!





Automated Tests must be

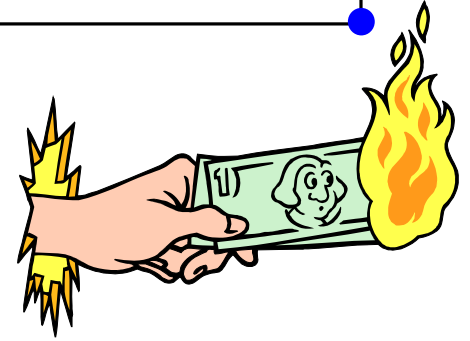
- easy to write
 - easy to find
 - easy to run
 - easy to maintain
- otherwise
- they will slow you down
 - they will get left behind
 - you'll go back to manual testing





Economy of Tests

- ❑ The number of possible defects is infinite
- ❑ Time and resources are usually finite ...



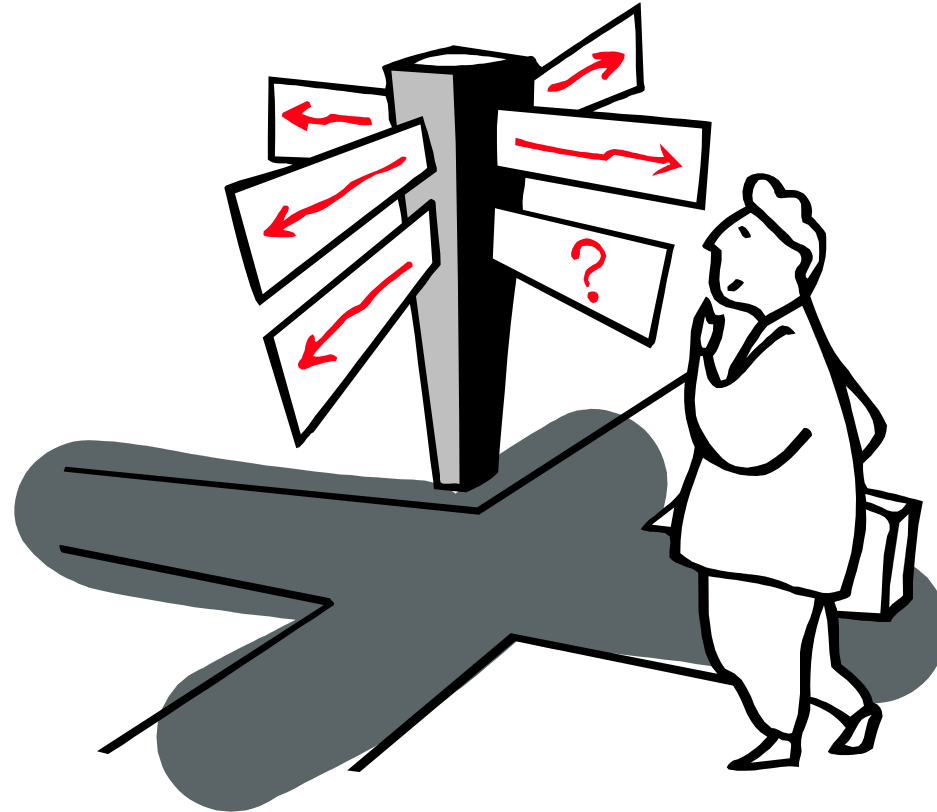
Hence every test strategy involves a trade off!

- ❑ Don't automate the overly complex cases (remember the 80-20 rule)
- ❑ Only automate tests that are going to be repeated
- ❑ Choose a methodology which maximizes both ease of writing and maintainability



Some Terminology ...

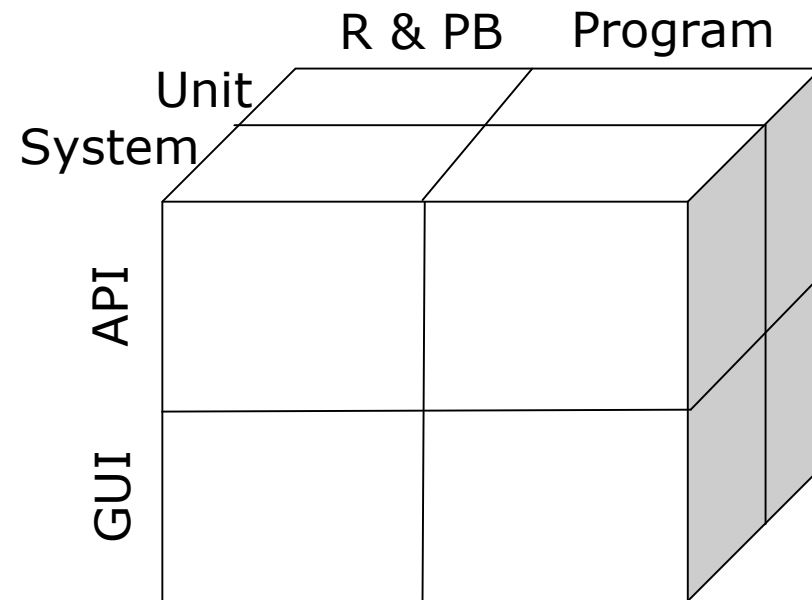
- Unit tests
- Integration tests
- System tests
- Acceptance tests
- Functional tests
- Black-box tests
- White-box tests
- ...





Classifying Automated Tests

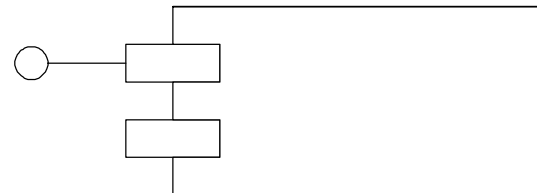
- Granularity
 - Entire system
 - Subsystems
 - Individual units
- Point of Contact
 - Existing User Interface
 - Testability API
- Test Case Production
 - Record and Play Back
 - Hand Written (programmatic)





Unit Tests

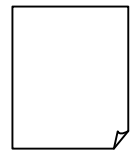
- ❑ Tests a logical unit in isolation
- ❑ Written and executed by developers as part of the development process
- ❑ Written in a programming language
- ❑ Once written, always must run 100% correct at all times before integrating any changes





Acceptance Tests

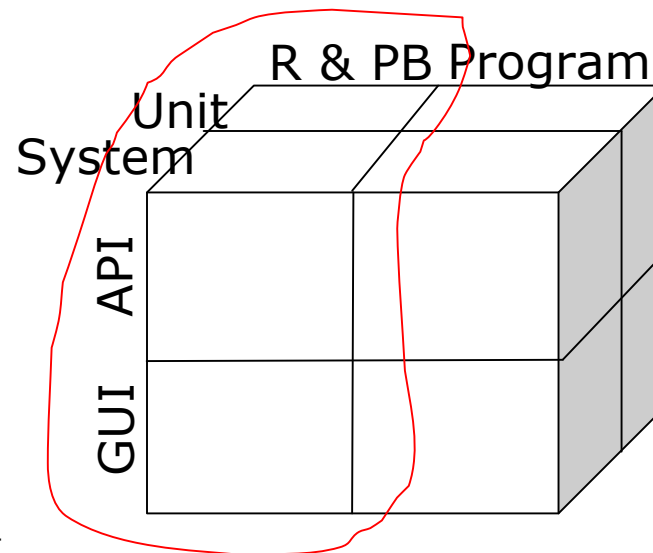
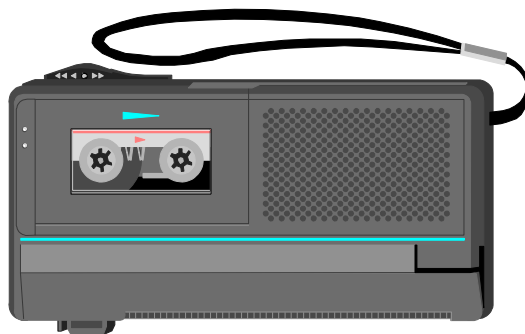
- ❑ Tests a system or part of a system from the outside, in terms of observable behavior
- ❑ Typically structured around a comprehensible chunk of system functionality
- ❑ Ideally written by the customer
- ❑ Either written in terms of User Interface actions, or in terms of “words” and “facts”
- ❑ Ideally run near 100% correct at the end of a release/project





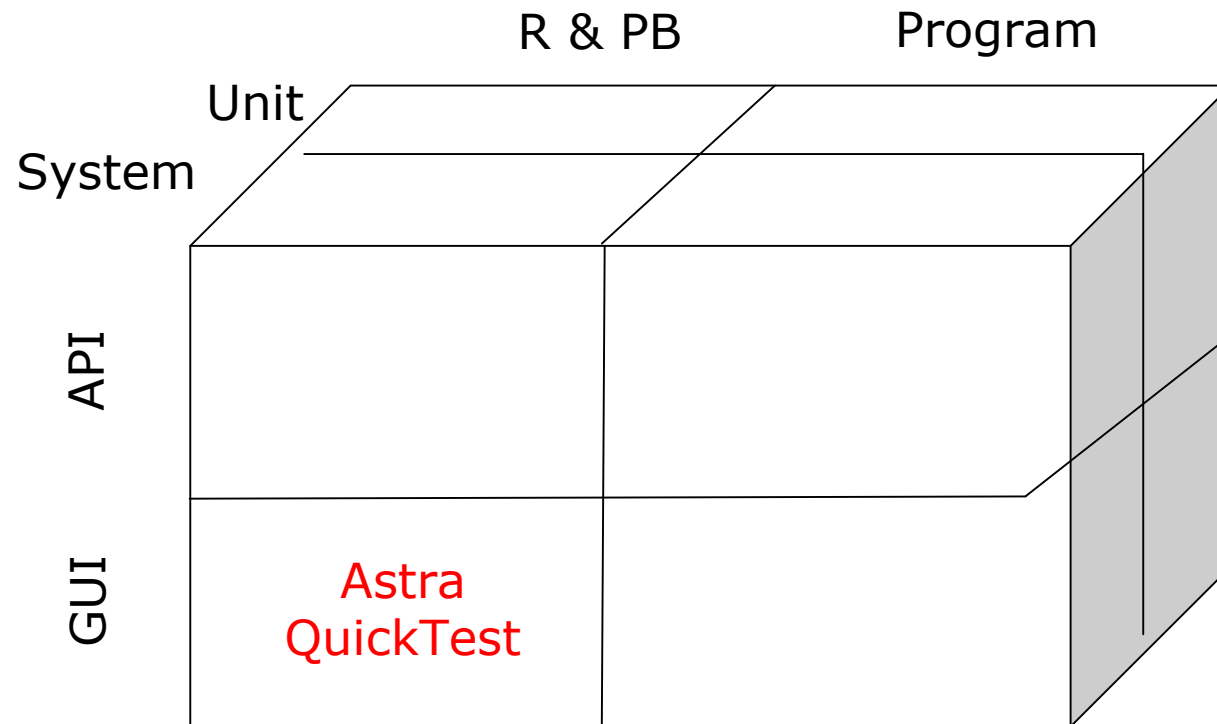
Record/Playback

- A Test Automation tool records “events” that make up a Test Case into a Test Script
- Events can be User Interface interactions or API calls
- The scripts can be played back later for regression testing



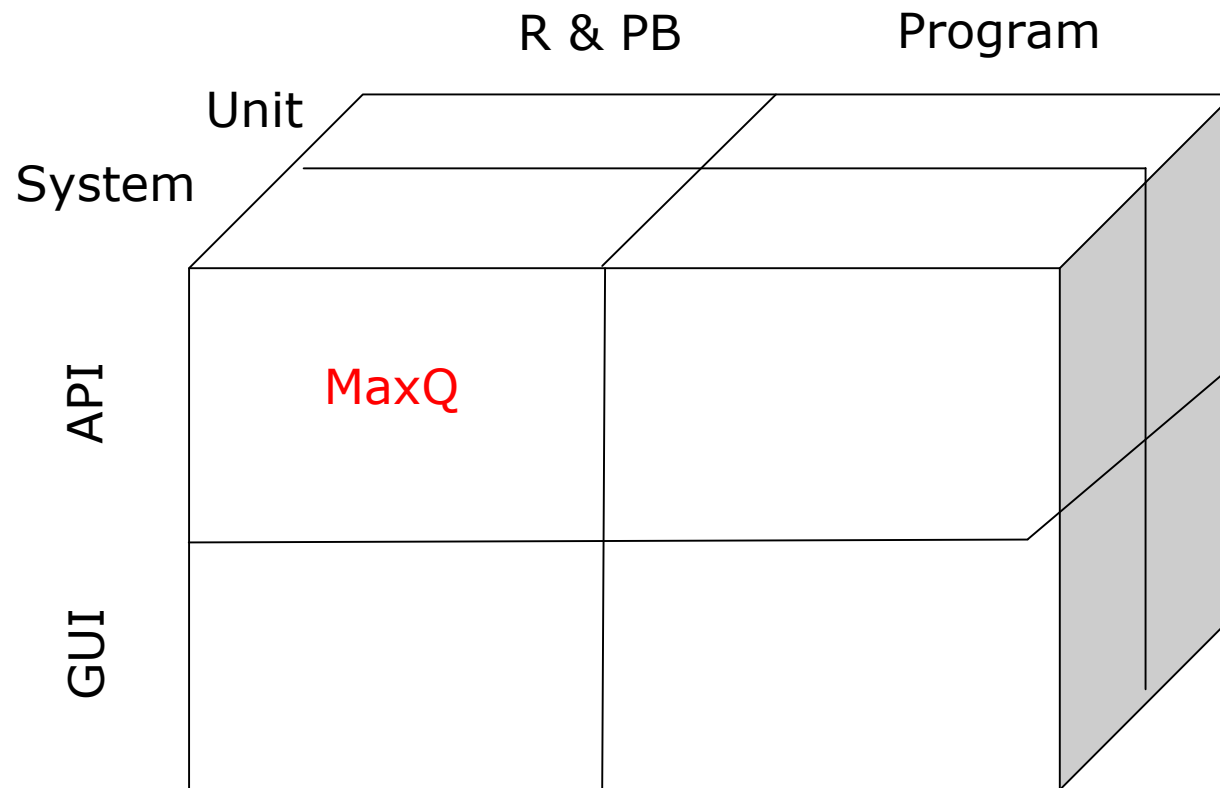


Example: Astra QuickTest





Example: MaxQ





Drawbacks of Record/Playback approaches

- Tests tend to be fragile
- Maintenance tends to be expensive
- Complex, expensive tools
- Tests cannot be pre-built





Hand-written, Programmatic Tests

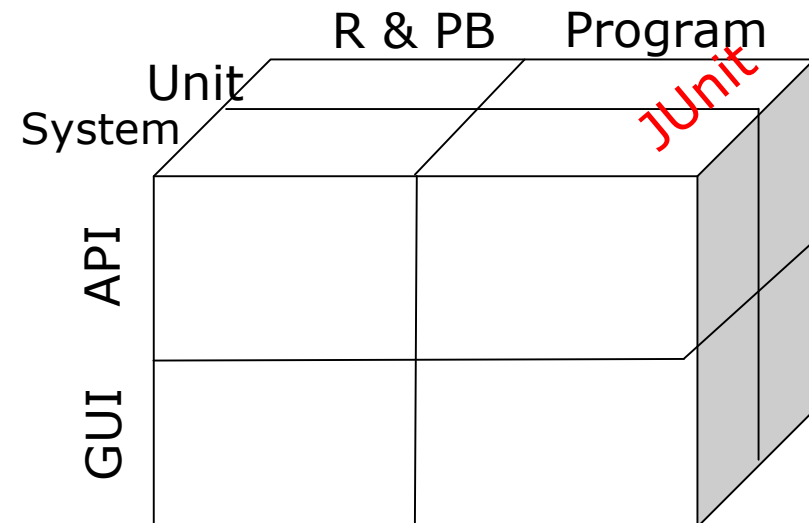
□ Pros:

- Tests can be made more robust
- Tests can be made more maintainable
- Simple, cheap tools and frameworks
- Can be built before production code



□ Cons:

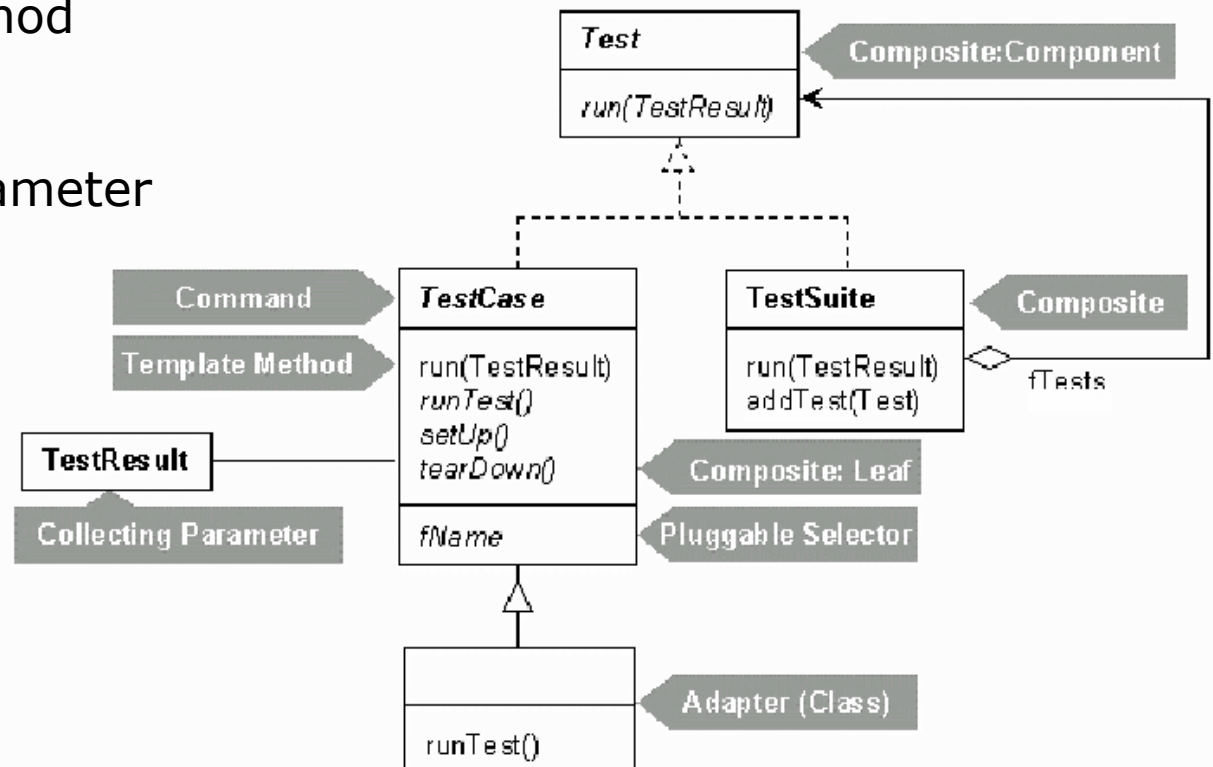
- Higher initial cost/effort
- More skills required





Leveraging JUnit beyond Unit Tests

- TestCase
 - Command
 - Template method
- TestResult
 - Collecting Parameter
- TestSuite
 - Composite





JUnit Add-ons for testing Web Applications

- www.HttpUnit.org
 - Framework which emulates browser behavior
 - form submission
 - JavaScript execution
 - http authentication
 - cookies
 - page redirections
 - Allows Java test code to examine and validate returned pages

- jWebUnit.sourceforge.net
 - higher level abstraction on top of HttpUnit that facilitates creation of acceptance tests for web applications



jWebUnit example

```
public class TestStore extends WebTestCase {  
  
    public void testLogin() throws Exception {  
        beginAt("/store/faces/home.jsp");  
        clickLinkWithText("Log in");  
        setFormElement("_id0:username", "test");  
        setFormElement("_id0:edtPassword", "test");  
        submit();  
  
        assertTextPresent("Welcome to Callista Store");  
        assertTextPresent("test");  
    }  
}
```



JUnit Add-ons for database testing

- www.dbUnit.org
 - JUnit extension targeted for database-driven projects
 - Can be used to put a database into a known state between test runs
 - Enables assertions to verify that the database content match some expected values.
 - Can export and import database content to and from XML datasets





dbUnit example

```
public class TestStore extends WebTestCase {

    public void setUp () throws Exception {
        IDataBaseConnection connection = getConnection();
        try
        {
            IDataSet dataSet = new FlatXmlDataSet(new FileInputStream(xmlFile));
            DatabaseOperation.CLEAN_INSERT.execute(connection, dataSet);
        }
        finally
        {
            connection.close();
        }
        getTestContext().setBaseUrl("http://localhost:8080");
    }

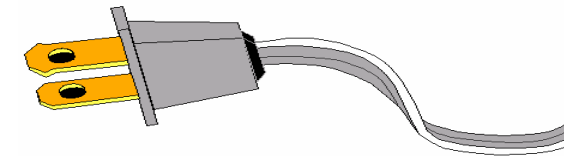
    private IDataBaseConnection getConnection() { ... }
}
```



Data-driven Acceptance Tests

- Tests written in terms of “words” (operations) and “facts” (data) that make sense to the customer
- Software modules adapt the “words” and “facts” to the underlying software system, interpret them and give them formal semantics

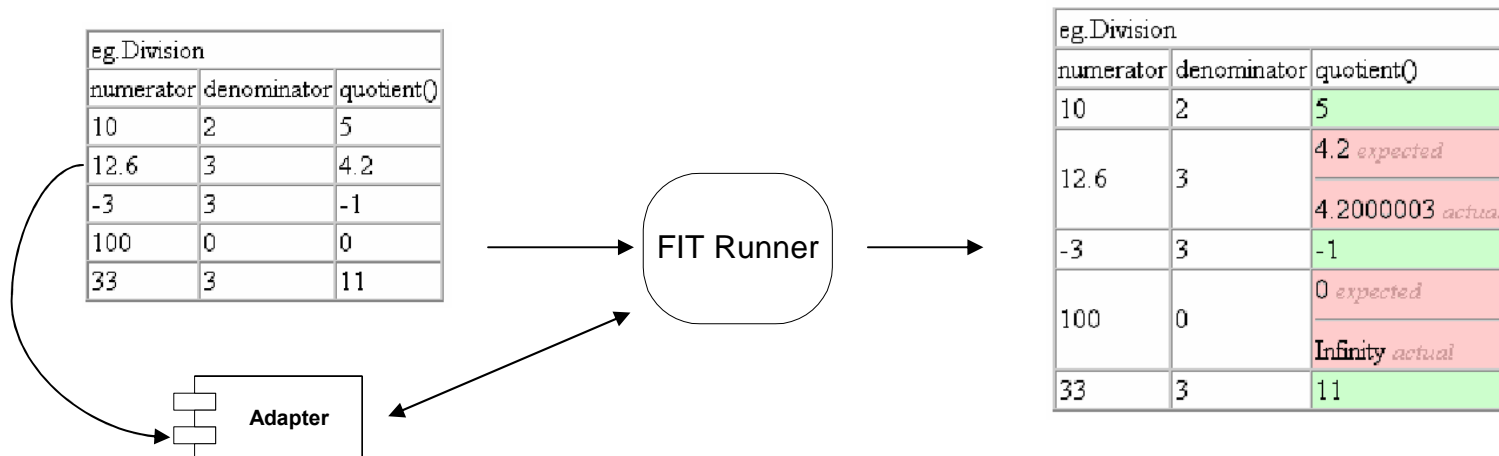
#	Column A Utility Keyword	Column B Field / Screen Name/Parameter	Column C Input/Verification Data Value	Column D Comment
1.	ENTER:	Payment_Amount Payment_Date Payment_Method_Check	125.87 [Pay_Date] Check	Post a Payment
2.	ACTION:	Press_Key	F9	Process Payment
3.	VERIFY:	Screen	Payment Screen	Verify screen remains
4.	VERIFY_DATA:	Payment Amount	125.87	Verify updated data
5.	END_TEST:	Press_Button	Done	Close Account window
6.		Verify_Window	Main Window	Return to main window





Framework for Integrated Test (FIT)

- Simplistic framework for data-driven tests
- Tests are viewed as documents containing tabular data
- Executing tests are viewed as annotating the test documents with test results



FIT Example



net - Microsoft Internet Explorer

Arktiv Redigera Visa Eavoriter Verktyg Hjälp

Bakåt - - - - - Sök Favoriter Media

Adress http://localhost:8080/store/fit_tests/fit_rmi.html

Acceptance Test, CallistaStore

Application Logic Layer tests

[Run test](#)

Setup Database

The database is setup with data from an [XML dataset definition](#).

se.callista.fit.DbUnitFixture	
driver	COM.ibm.db2.jdbc.app.DB2Driver
url	jdbc:db2:Sample
user	db2admin
passwd	default
xmlFile	D:\tmp\CallistaStore\CallistaStoreAp
clean insert	

Verify Product Catalog content

Klar

net - Microsoft Internet Explorer

Arktiv Redigera Visa Eavoriter Verktyg Hjälp

Bakåt - - - - - Sök Favoriter Media

Adress http://localhost:8080/store/fit_tests/fit_with_jWebUnit.html

Log on

net.sourceforge.jwebunit.fit.WebFixture		
baseUrl	http://localhost:8080	
begin	/store/faces/home.jsp	
press	link	Log in
enter	_id0:username	test
enter	_id0:edtPassword	test
press	submit	
check	text present	Welcome to Callista Store
check	text present	test

View Product Catalog

net.sourceforge.jwebunit.fit.WebFixture		
press	link	Product Catalog
check	text present	100
check	text present	Dummy Product 1
check	text present	150
check	text present	Dummy Product 2

Klar Lokalt intranät



Finally – don't forget the people!

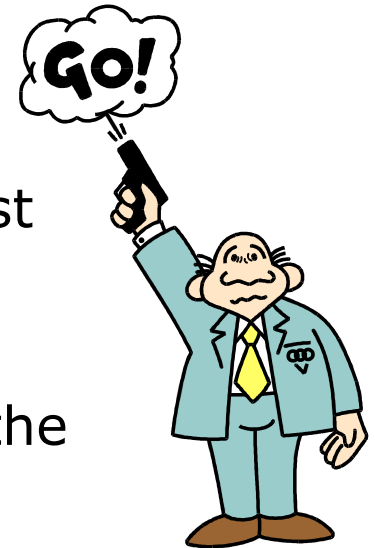
- Managing Resistance to Change
 - The tool will replace the testers?
 - The tool will be too expensive?
 - The tool will be too difficult to use?
- QA team ♥ Development Team = true?
 - Co-operative relationship
 - Open to learning
- Management Buy-In
 - Early Payback





Summary

- ❑ Establish clear and reasonable expectations as to what can and cannot be accomplished with automated testing
- ❑ Adopt a viable, cost-effective test strategy and methodology
- ❑ Select tools and frameworks that allows you to automate your tests in accordance with your test strategy
- ❑ Gain approval for the test strategy both within the project team and from the project sponsor





How can Callista help you?

- ❑ Seminars, Workshops and hands-on tutorials
- ❑ Initial tool smithing to establish a lightweight, efficient testing platform
- ❑ Architectural consulting to assure testability
- ❑ Technical management and architectural mentoring for Unit-Test and Acceptance Test patterns
- ❑ Project architects, test engineers and mentors to secure a successful testing strategy



Resources

- Unit Testing and Unit Testing Extensions
 - www.junit.org
 - www.httpunit.org
 - jWebUnit.sourceforge.net
 - www.dbUnit.org
 - jfcUnit.sourceforge.net

- GUI Testing
 - www.mercuryinteractive.com
 - MaxQ.tigris.org

- FIT
 - fit.c2.com



Questions

