Home-Brew Test Automation

bret@pettichord.com
www.pettichord.com

Software Testing Australia New Zealand, November 2003

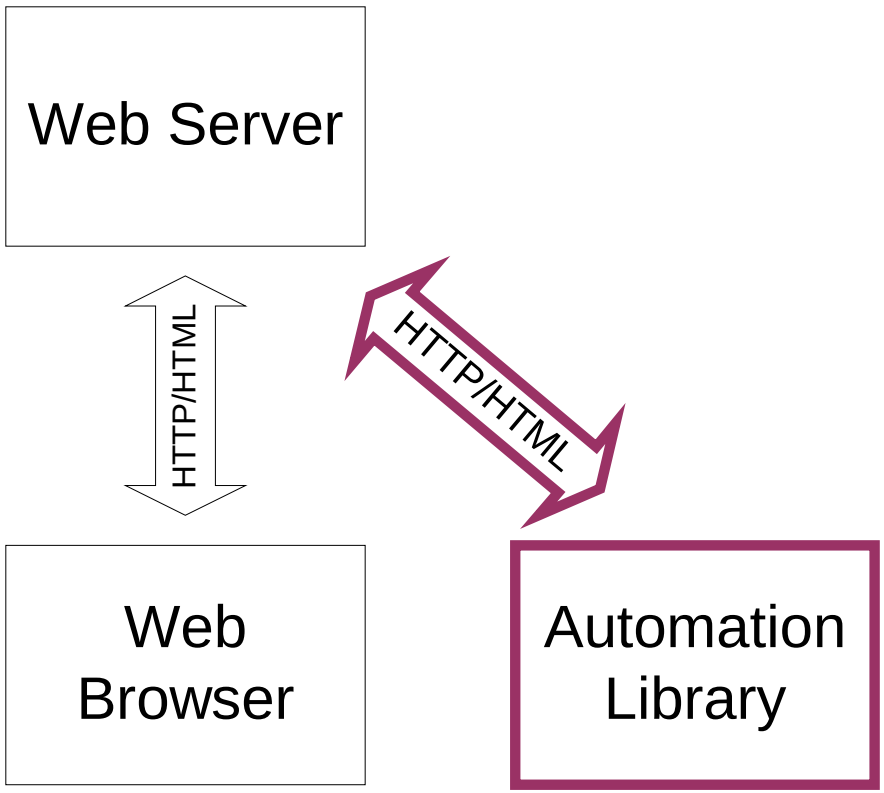
XP and Automated Testing

- Programmers write automated unit tests.
- Acceptance tests must also be automated.
- Programmers and testers *work together* on acceptance tests.

XP Teams Rarely Use Commercial GUI Test Tools

- Objections to Commercial Tools
  - Price
    - Everyone on the team needs to be able to run the tests.
    - Can’t afford to give copies to everyone
  - Tool Languages
    - Understood by few
    - Often weak and limited: “Heinous”
- Often prefer building their own testing frameworks

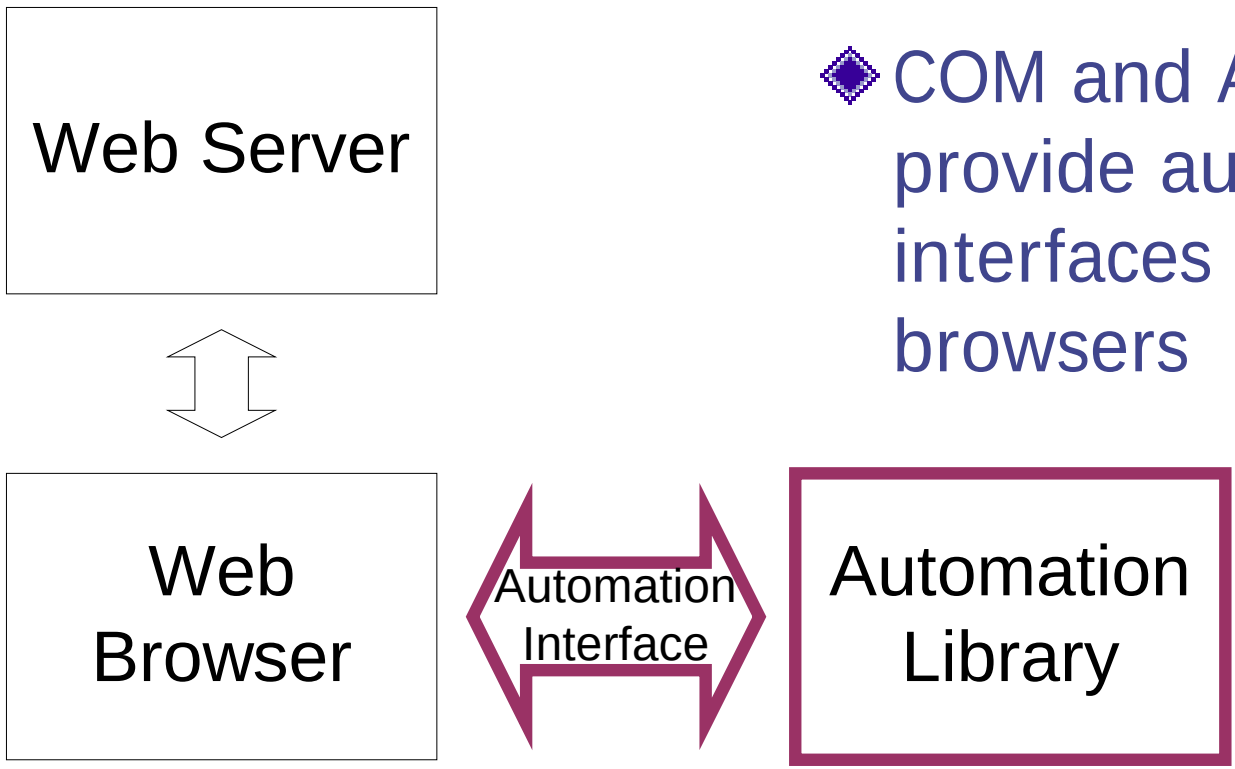
Example: Browser Simulation



- HTTP-Unit
  - Java
  - Perl
  - Ruby
  - Python

Tests execute directly against the web server

Example: Browser Automation



- COM and Applescript provide automation interfaces to browsers

Tests execute against a browser

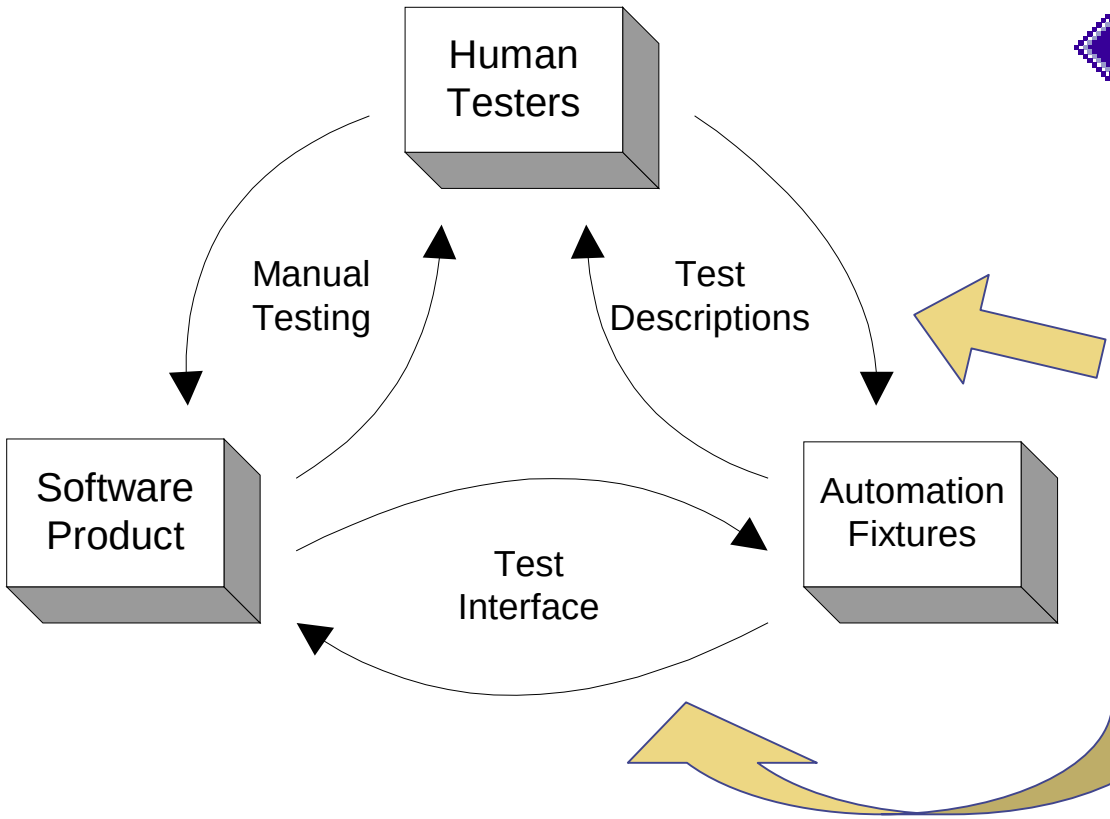
Home-Brew Strategies

1. Extending Unit Testing
2. Adapting the Product
   - Thin GUI
   - Test Interfaces
3. Building Your Own Tool

These strategies are:

- *Used by XP teams*
- *Available to you*
- *More effective when combined*

Test Interaction Model



- Customizing Testing Frameworks
  - For tester usability
  - For product compatibility

Agenda

- Beyond Unit Testing
- Scripting Languages
- Interface Drivers
- Building Your Own Tool
- Test Description Languages

  Break

- Web Browser Testing in Ruby

✓ Beyond Unit Testing
Scripting Languages
Interface Drivers
Building Your Own Tool
Test Description Languages

Beyond Unit Testing

- XP has made programmers *love* unit testing
  - JUnit has been:
    - ported to dozens of languages
    - extended for dozens of frameworks
    - incorporated in dozens of IDEs
  - Developers all over are now writing unit tests
- XP leaders are now building tools for acceptance testing...

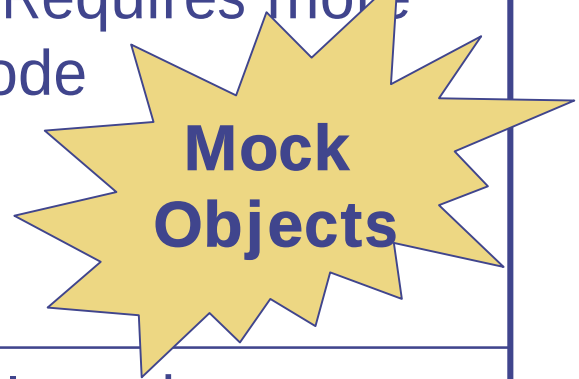
What is Unit Testing?

- Units are functions, methods or small bits of code, usually written by a single programmer.
- Unit tests are written in the same language as the code being tested.
- Unit tests are written by the programmers who wrote the the code being tested.
- A test harness or framework collects tests into suites and allows them to be run as a batch.

Unit Integration Testing

- How to test units that depend on other units?

Unit isolation testing *Test each unit in isolation*	*Create stubs and drivers objects for external units*	Requires more code
Unit integration testing *Test units in context*	*Call external units*	Introduces dependencies. Test suites take longer to run



Test-Driven Development

- Developers write unit tests *before* coding.
  - Motivates coding
  - Improves design
    - reducing coupling
    - improving cohesion
  - Provides regression tests
- An approach to design
  - More than just as test strategy
  - Specification by Example
  - Refactoring

```
public void testMultiplication() {
  Dollar five = Money.dollar(5);
  assertEqual(new Dollar(10), five.times(2));
  assertEqual(new Dollar(15), five.times(3));
}
```

Home-Brew Ingredients

1. Test Harness
   - Collecting tests so they can be executed together
2. Language
   - Creating the automation fixtures
   - Providing a language for describing tests

   (These may be the same or may differ)
3. Product Interface Driver
   - Giving access to the software product

These are the ingredients of any *automated testing system.*
This talk will discuss languages and drivers.

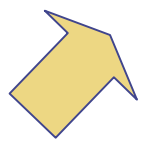
Test Harness

Necessary Capabilities

- Run many test scripts
- Collect test verdicts (pass or fail)
- Report test results

If you don' t have this, you don't have a test harness

Depending on your circumstances, you may find many of these other capabilities to be necessary.



Additional Capabilities

- Check test preconditions (abort or correct if not met)
- Allow selected subsets of tests to run
- Distribute test execution across multiple machines
- Distinguished between known failures and new failures
- Allow remote execution and monitoring
- Use Error Recovery System (later)

Open Source Harnesses

	Character-based testing	Unit testing	Command-line testing
Interface Driver	**Expect**	N/A	N/A
Language	TCL	Java	Perl, Shell
Test Harness	**DejaGNU**	**JUnit**, etc	**TET**

Three Kinds of Languages

- System Programming Languages
  - Optimized for *performance*.
  - What your programmers are probably using.
  - **C, C++, Java, C#**
- Scripting Languages
  - Optimized for *ease of use* and *high productivity*.
  - Command interpreters facilitate learning and exploration.
  - **Perl, Tcl, Python, Ruby, VBScript, JavaScript, Rexx, Lua**
- Data Presentation Languages
  - Optimized for *readability* and *structure*.
  - No logic
  - **HTML, XML, CSV, Excel**

What Many Testers Use Today

```
public function stack_init (inout stack[]) {
   auto tmp;
   for (tmp in stack) delete stack[tmp];
   stack["next"] = 0;
   return E_OK;
}
public function stack_push (inout stack[], entry) {
   stack[stack["next"]++] = entry;
   return E_OK;
}
public function stack_pop (inout stack[], out out_entry) {
   auto res = E_OK;
   if (stack["next"] < 1) {
        res = E_OUT_OF_RANGE;
   } else {
        out_entry = stack[stack["next"] - 1];
        delete stack[stack["next"] - 1];
        stack["next"]--;
   }
   return res;
}
```

This code implements a stack.

Source: “Breaking the Language Barrier,” Meisenzahl and Firmansjah

Scripting Languages for Testing

Beyond Unit Testing
✓ Scripting Languages
Interface Drivers
Building Your Own Tool
Test Description Languages

- **Perl**
  - Well-established
  - Vast **libraries**
- **Tcl**
  - Well-established
  - Compact
  - Popular with **embedded** systems
- **Python**
  - Concise support for object-oriented programming
  - Integrates well with **Java** (Jython)

- Everything's an object
- Principle of **least surprise**

- **Visual Basic & VB Script**
  - Popular
  - Integrate well with **Microsoft** technologies

- Tcl, Python and Ruby have *command interpreters*
- All but VB are *open source*
- All are well supported

The best language is the one your team knows.

Language Choices

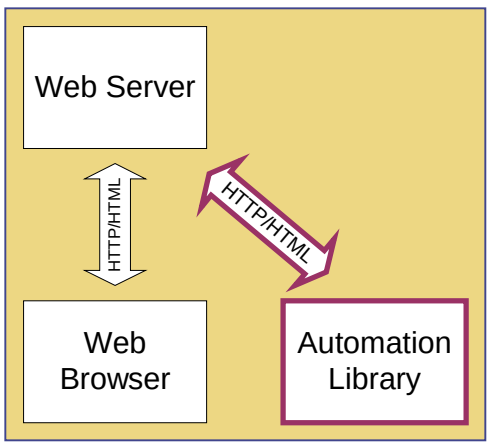
What should you write tests in?

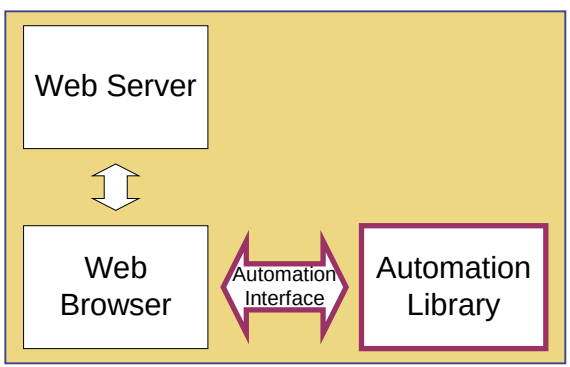
1. **System programming language**
   - Reuse unit test harness
   - May result in lower productivity
2. **Scripting language**
   - Requires interface to product
   - Allows most kinds of tests
3. **Data presentation language**
   - Requires support code in a scripting or system language
   - May improve understandability

- Options
  - 1 only
  - 2 only
  - 1 and 3
  - 2 and 3
  - All three?

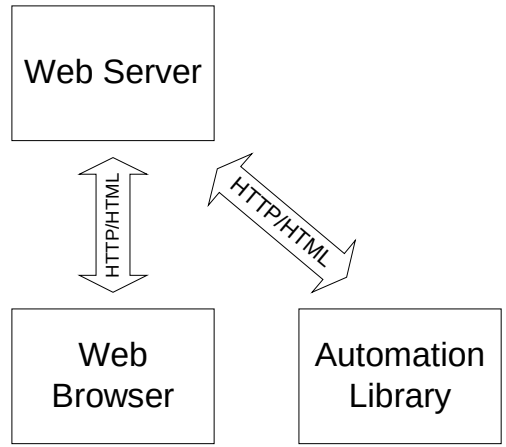
Interface Drivers

- How do your tests access the product?
  - Simulation
    - Access the server in the same way as the client or browser.
    - Use Thin GUI to minimize the untested code.
  - Automation
    - Automate the client or browser using automation interfaces.





Open Source Browser Simulation



- **HttpUnit**, Russell Gold
  - Browser simulation in Java. Popular & well-extended.
  - http://www.httpunit.org/
- **jWebUnit**, Thoughtworks
  - A refinement on HttpUnit and FIT. Java-based.
  - http://jwebunit.sourceforge.net
- **Canoo WebTest**, Canoo Engineering AG
  - Java-based browser simulation with XML.
  - http://webtest.canoo.com
- **TestMaker**, Frank Cohen
  - Python test scripts, Java-based tool. Also simulates non-browser clients.
  - http://pushtotest.com
- **HTTP::WebTest**, Richard Anderson and Ilya Martynov
  - Browser simulation in Perl.
  - http://search.cpan.org/author/ILYAM/HTTP-WebTest-2.00/
- **WebUnit**, Yuichi Takahashi
  - Browser simulation in Ruby.
  - http://www.xpenguin.biz/download/webunit/index-en.html
- **Puffin**, Keyton Weissinger
  - Browser simulation in Python and XML.
  - http://www.puffinhome.org/

More Tools

- http://www.junit.org/news/extension/web/index.htm

HttpUnit Example

```
public void testLoginSuccess() throws Exception {
    WebConversation conversation = new WebConversation();
    String url = "http://localhost:8080/shopping/shop“;
    WebResponse response = conversation.getResponse(url);
    assertEquals(“Login”, response.getTitle());

    WebForm form = response.getFormWithName("loginForm");
    WebRequest loginRequest = form.getRequest();
    loginRequest.setParameter("user", “mike”);
    loginRequest.setParameter("pass", “abracadabra”);
    response = conversation.getResponse(loginRequest);
    assertEquals(“Product Catalog”, response.getTitle());
}
```

Example Courtesy of Mike Clark

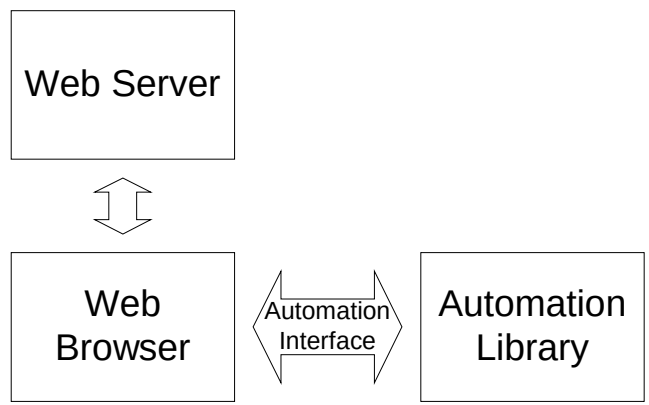
Canoo WebTest Example

```
<project name=“ShoppingCartTests" default=“main”>
    <target name="main">
        <testSpec name=“loginSuccessTest">
            <config host="localhost“ port=“8080“
                protocol="http“ basepath=“shopping” />
            <steps>
                <invoke url=“shop" />
                <verifytitle text="Login" />
                <setinputfield name="user“ value="mike" />
                <setinputfield name="pass“ value=“abracadabra" />
                <clickbutton label="Login" />
                <verifytitle text=“Product Catalog" />
            </steps>
        </testSpec>
    </target>
</project>
```

Example Courtesy of Mike Clark

Open Source Browser Automation

- **Cliecontroller**, Chris Morris
  - IE and .Net Windows forms automation in Ruby
  - http://www.rubygarden.org/ruby?IeController
- **Samie**, Henry Wasserman
  - IE Browser automation in Perl
  - http://samie.sourceforge.net/



Open Source Java GUI Drivers

- **Marathon**, Jeremy Stell-Smith et al, Thoughtworks
  - Java Swing GUI driver using Python scripts. Includes a recorder.
  - http://marathonman.sourceforge.net/
- **Abbot**, Timothy Wall
  - Java GUI driver and recorder using XML scripts.
  - http://abbot.sourceforge.net/
- **Pounder**, Matthew Pekar
  - Java GUI driver and recorder.
  - http://pounder.sourceforge.net/
- **Jemmy**
  - Java GUI driver. Integrated with NetBeans.
  - http://jemmy.netbeans.org/
- *More Tools*
  - http://www.junit.org/news/extension/gui/index.htm
  - http://www.superlinksoftware.com/cgi-bin/jugwiki.pl?TestingGUIs

Free Windows GUI Drivers

- **Win32-GuiTest**, Ernesto Guisado
  - Perl Library. Popular. Strong support for various controls.
  - Open Source
  - http://search.cpan.org/author/ERNGUI/Win32-GuiTest-1.3/
- **Win32-CtrlGUI**, Toby Everett
  - Perl library. Strong support for window identification.
  - Open source
  - http://search.cpan.org/author/TEVERETT/Win32-CtrlGUI-0.30/
- **Novell AppTester**
  - API typically called from C++.
  - Distributed as binary as part of Novell's system testing tools.
  - http://developer.novell.com/ndk/softtestv3.htm
- **Bugslayer Tester**, John Robbins
  - Recorder and library written in VB and C++. Well-documented.
  - Provides COM interface supporting VBScript & Jscript.
  - Source and executables published on MSDN.
  - http://msdn.microsoft.com/msdnmag/issues/02/03/bugslayer/default.aspx
- **AutoIt**
  - Recorder
  - Library delivered as ActiveX component (VBScript)
  - Free download, freely distributable. Closed source.
  - http://www.hiddensoft.com/AutoIt/

Other Open Source Test Libraries

- **Expect**, Don Libes
  - Command line driver in TCL. Long-established.
  - http://expect.nist.gov/
- **Framework for Integrated Test** (FIT), Ward Cunningham
  - Parses tests in HTML. Supports multiple languages.
  - http://fit.c2.com
- **Android**, Larry Smith
  - Unix/Linux GUI driver in Tcl.
  - http://www.wildopensource.com/larry-projects/android.html
- **OpenSTA**, Cyrano
  - Web performance testing driver and recorder. Uses a "proprietary" scripting language.
  - http://opensta.org/
- **STAF**, Charles Rankin, IBM
  - Distributed multi-platform testing framework
  - http://staf.sourceforge.net/index.php

More Open Source Tools

- Tool Listings
  - Opensourcetesting.org
  - Xprogramming.com/software.htm
  - Junit.org/news/extensions
- *Open Testware Reviews*
  - Monthly Newsletter by Danny Faught
  - Tejasconsulting.com/open-testware

Building Your Own Tool

Beyond Unit Testing
Scripting Languages
Interface Drivers
✓ Building Your Own Tool
Test Description Languages

- It's now easier than ever
  - Use standardized automation interfaces
  - Only support one interface technology
  - Reuse test harnesses and languages
  - Use and extend open source interface drivers

Home-Brew Ingredients

1. Test Harness
2. Language
3. Product Interface Driver

Approaches

- Scripting *Easiest*
- Data-driven
- Capture/Replay *Hardest*

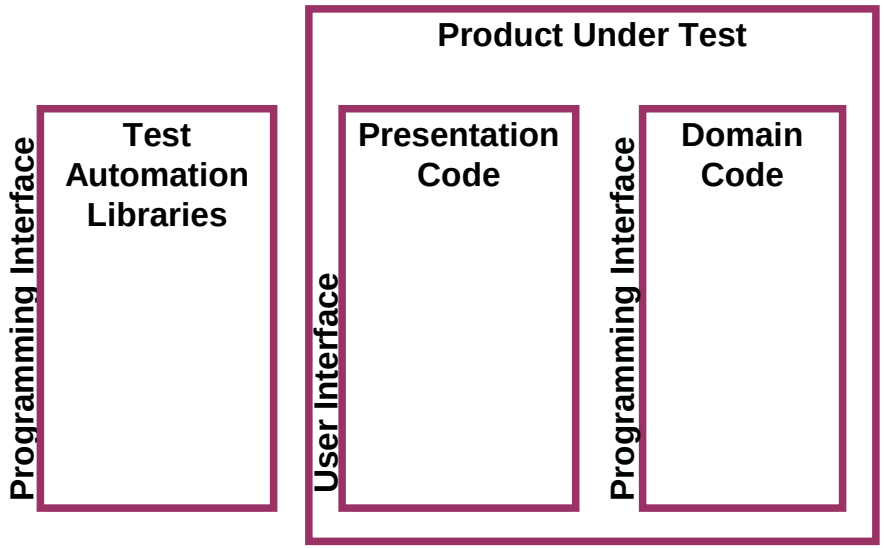
Build a Tool or Adapt the Product?

- Choosing the interfaces will you use for testing is a key strategic decision.
  - **Build a Tool.** Add fixtures and tools to access existing interfaces; or
  - **Adapt the Product.** Expose or create interfaces for direct testing of the product.
  - *This difference is actually rather moot with test-driven development!*
- A sound approach requires close cooperation and trust between testers and developers.

Adapting Your Product

Use Existing Interfaces

- Products using existing APIs for testing
  - InstallShield
  - Autocad
  - Interleaf
  - Tivoli
- Web Services Interfaces are ideal!



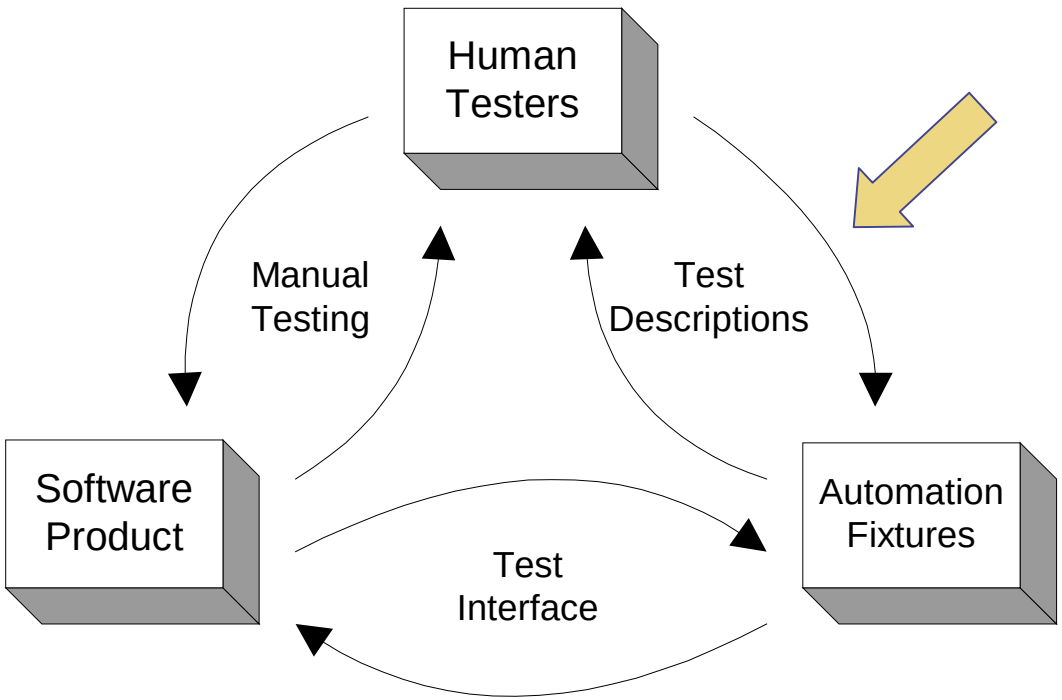
Create New Interfaces

- Products exposing interfaces specifically for testing
  - Excel
  - Xconq (a game!)

Test interfaces provide control and visibility

Test Description Languages

Beyond Unit Testing
Scripting Languages
Interface Drivers
Building Your Own Tool
✓ Test Description Languages
Coaching Tests



Providing an effective means for describing tests which...

- Testers can create.
- Fixtures can execute automatically.
- Anyone can understand.

Test Description Languages

What is the best test description language for expressing tests?

- Tables
  - Often readable to more people
  - Require fixtures & parsers
- Scripts
  - Better support for variables and looping
  - Require less fixturing

```
Timeclock> start 'misc'
Timeclock> pause
Timeclock> start 'stqe'
Timeclock> jobs
misc, started 02002/08/30 4:32 PM, is paused.
stqe, started 02002/08/30 4:33 PM, is recording time.
```

Example courtesy Brian Marick

FIT tests

- Scrape tests from HTML docs
- Keep requirements & tests together
- Check automatically
- Browse results online
- Understandable by everyone

Division shall work with positive and negative numbers.

eg.Division		
numerator	denominator	quotient()
1000	10	100.0000
-1000	10	-100.0000
1000	7	142.8571 *expected* 142.85715 *actual*
1000	.001	1000000 *expected* 999999.94 *actual*
4195835	3145729	1.3338196

Beyond Unit Testing
Scripting Languages
Interface Drivers
Building Your Own Tool
Test Description Languages
✓ Coaching Tests

Coaching Tests

- Use tests to drive development
- Tests provide:
  - Goals and guidance
  - Instant feedback
  - Progress measurement
  - Health check of the project
- Tests are specified in a format:
  - Clear – so any one can understand
  - Specific – so it can be executed

eg.ArithmeticFixture					
x	y	x + y	x - y	x * y	x / y
200	300	500	-100	60000	0
400		420	380	8000	20

Test Automation in the Silo

- Traditionally test automators have worked in a separate space from developers
  - The code is separate
  - The teams are separate
  - *Ex post facto* GUI automation
- Reasons for change
  - Tool/application compatibility (testability)
  - Maintenance when GUI changes
  - Testing needs to be everyone's concern

You can change whether you are are using XP or not!