ThoughtWorks
The art of heavy lifting

Homebrew Test Automation

bret@pettichord.com
www.pettichord.com

A One-Day Seminar

September 2004

About Bret Pettichord

Bret Pettichord, a software testing expert and an influential author and speaker, joined ThoughtWorks in July 2004. Mr. Pettichord serves ThoughtWorks, Inc. as a test architect, implementing effective technologies for automated testing and promoting responsible methodologies for agile testing and quality assurance.

His software testing philosophy is context-driven, focusing on uncovering important risks, maintaining close relations with programmers, and using agile testing methods that provide rapid feedback. He has broad experience using commercial and open-source tools for automated testing.

Mr. Pettichord is a founder of the Context-Driven School of software testing, which sees testing as a technical investigation of software risk that requires skill, adaptability and tact. He co-authored *Lessons Learned in Software Testing* (a Jolt Award finalist) to explain the thinking of the School. He has published over two dozen papers on software testing and test automation. His ideas about homebrew automation, agile testing and testability have been featured in *Application Development Trends* and *The Rational Edge.*

As a member of the Agile community, he has regularly hosted workshops that have brought together leading testers and programmers to assess and develop methods for testing on agile projects. Mr. Pettichord founded the Austin Workshop on Test Automation in 2000. It’s a yearly event that brings together leading test automators. He has been regularly contributing to similar workshops since the first meeting of the Los Altos Workshop on Software Testing in 1996. He regularly speaks at conferences around the world.

A Homebrew Mishap

- In early 1990's, I contributed to a homebrew test automation system.
- With a mixed record of success, I was laid-off in 1993.
- I went to work for a test tool vendor and believed that commercial tools were the best way to avoid pitfalls.
- For many years thereafter, I was a tool advocate at large software companies.
- But I've changed my mind. Why?

3

One of my first test automation projects was on a home-brew automation tool at Interleaf. We had a small team of some four people working to build a tool for automating GUI testing. This was a decade ago, just before commercial GUI testing tools became available. We had some smart ideas and we also got caught up in some rat holes. Our progress was slow.

This was during the last recession and our company had layoffs that affected both me and most of the automation team. We were infrastructure, which is never a good place to be when layoffs come.

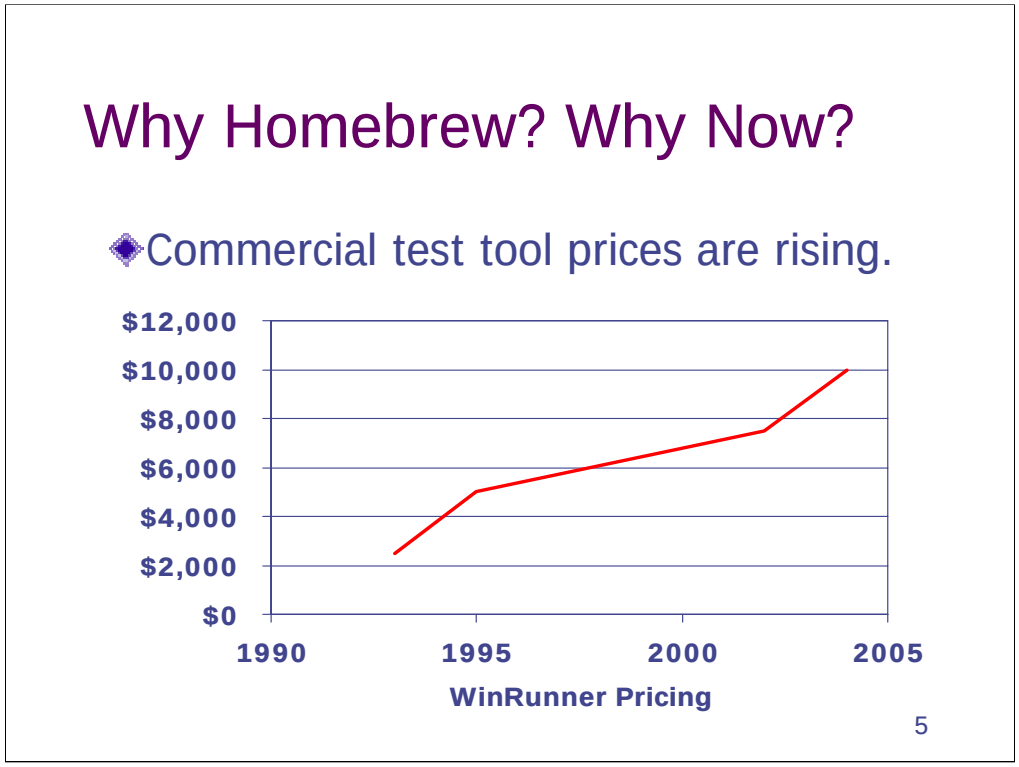
My next job was with a tool vendor, Segue, and the lesson that i learned then was that you shouldn't build a tool when you could purchase one. I later spent many years as a tool champion at BMC Software and IBM/Tivoli.

But i am here today to talk to you about teams that more recently have been building their own tools and finding success. I've heard lots of reports.

And building homebrew test automation systems has become the core of my consulting practice. It works.

Why Homebrew? Why Now?

- The industry now generally understands how to use test tools.
  - Four years ago, my “capture/replay is foolish” talk was controversial.
  - Scripting, Data-driven, and Keyword approaches are now accepted.



In general, tools for software development have been getting cheaper and adding more features over the past 10 years. Test tools have bucked this trend.

Why Homebrew? Why Now?

- Developers want to help.
  - Many have recently learned to automate unit testing.
  - Many are now motivated to support automated system testing.
  - Many are “test-infected.”
  - Tight job market increases their interest in delivering quality code.

Why Homebrew? Why Now?

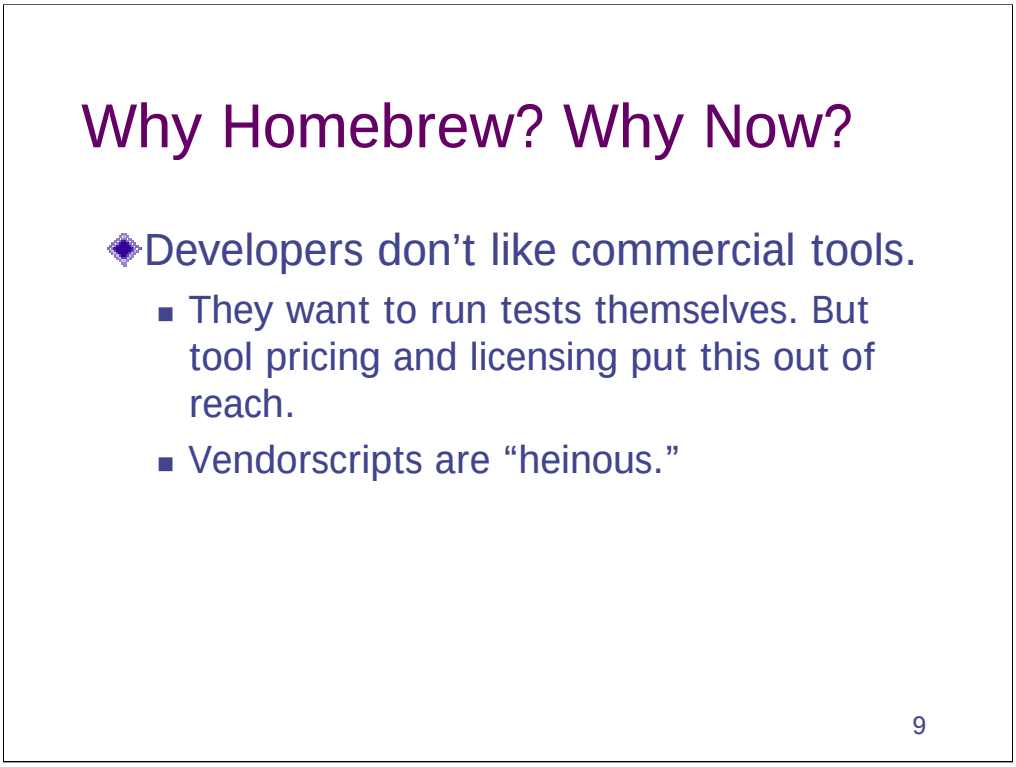
- Test tools are easier to build.
  - Software technology is now more testable.
    - COM, XML, HTTP, HTML and other standardized interfaces
    - *Reflection* makes it easier to find and attach to objects (Java & .Net)

7

Why Homebrew? Why Now?

- Many open-source frameworks are already available.
  - Growing interest in open source in general.
  - Kent Beck popularized test-driven development by making JUnit open-source.
  - Test-infected developers establish reputations by porting or extending xUnit.

8



XP developers don’t often use commercial GUI test tools. I've consulted with some teams and have talked to many others at conferences and workshops. Most have considered GUI test tools: they've tried them out and decided that they weren't for them.

There are two reasons.

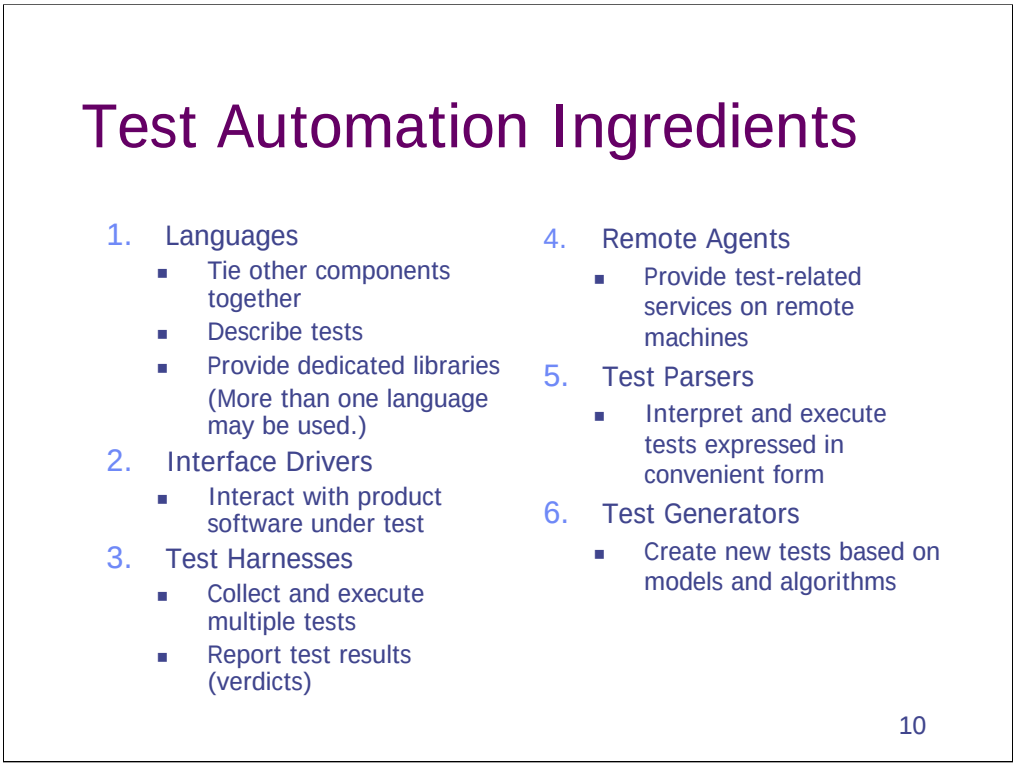
One is simply price. The standard industry practice of buying test tools for just the testers is unacceptable with XP. The point of automated tests is to allow anyone to run them anytime they need to. Which is often. So they need to be able to be run on anyone’s machine. It’s a key tool, like an IDE.

These days licenses for GUI test tools often run to up to $7,500 a seat. That's more than most IDE’s. It's hard enough for traditional organizations to find the budget to buy some licenses for their testers. XP groups, however, have to consider buying copies for everyone on the team. With costs like that you can see why they take a good look at alternatives.

For a long time, testers have been wishing that the tool vendors would supply run-only versions at a discount. I've never quite understood why there's been so little interest in providing these.

The other reason is that programmers find the languages used by these tools distasteful. Most use proprietary languages that are idiosyncratic and weak. Because XP teams make everyone responsible for testing, they need to write tests in a language that everyone can understand. And they want to use a language with modern features so that they can use the kinds of programming techniques that they are used to. Most proprietary "vendorscripts“ don't support object-oriented programming or pointers or exception handling. They've been called "heinous."

This view contrasts with a recent report by the Gartner Group that says that GUI Test Tools have reached a plateau of industry stability.



These are the ingredients of any test automation system or framework, commercial, open-source, or homebrew.

Most commercial testing tools include most of these ingredients. Their users may not be aware of the separate components. It's all part of one tool.

Some open-source tools also combine features that belong in more than one category. Even when this happens, the separate ingredients may have separate authors and names. So it's easier to be aware of the ingredients being used.

When you're building your own homebrew testing framework, you will likely put together tools from more than one category.

Any functional testing system will require a language, an interface driver and a harness, at a minimum. You may also wish to include other ingredients.

Homebrew usually means open-source, but not always. **If a tool isn't open-source, it will be noted. Otherwise you can assume it is.**

Terminology

Interface drivers are sometimes called *test drivers*. Test harnesses are also sometimes called test drivers. (That's why I avoid the term *test driver*.)

The word framework is often used to describe testing tools. It's a pretty broad and flexible term. I use it in a general sense to describe the collection of components that support your tests. Your framework may be made out of components that describe themselves as frameworks.

There is a lot of material to cover in this one-day seminar. Nonetheless, my seminars work best when we get questions and discussion. Your questions and interests are very important.

It's not easy sitting in one place all day. Feel free to stand up and walk around if it will help you. I do it all the time when I attend seminars.

I like to take breaks every hour. You and I both need it. If it seems like it should be time for a break and I'm droning on, please let me know. You're probably right.

Right now you are probably thinking, when are we going to eat lunch?

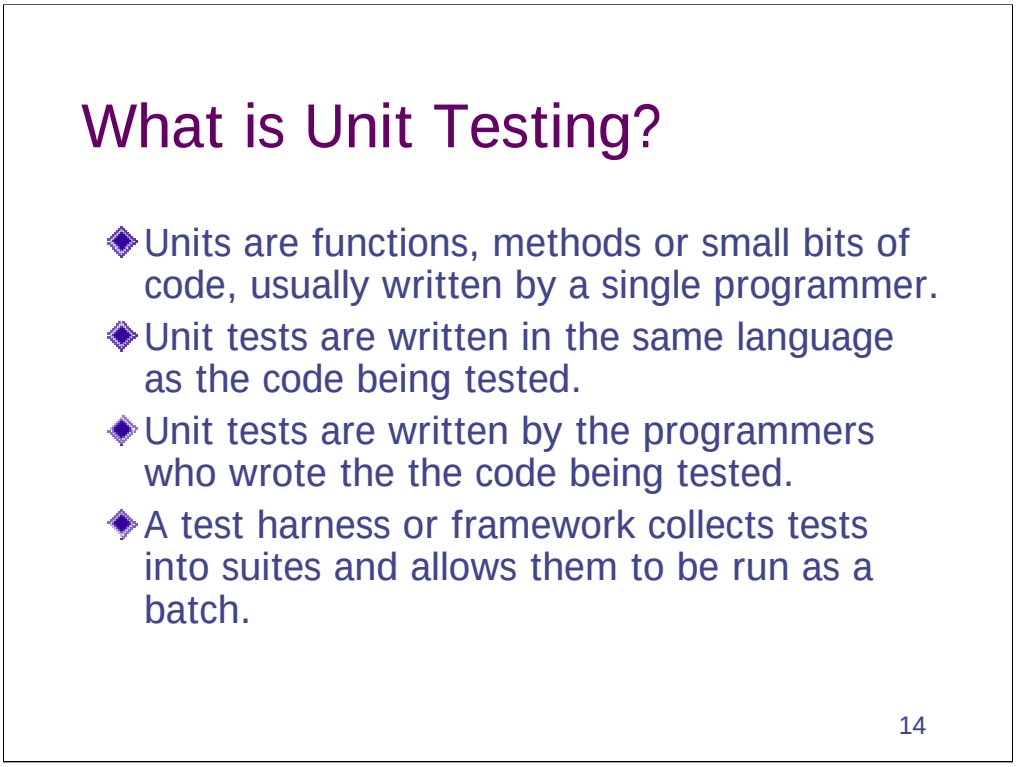
Welcome to this seminar. Thanks for coming.

We think this is an important topic and want you to be able to get the most out of this seminar. Thus we offer the following suggestions.

- Feel free to **introduce yourself** before the seminar or during a break. Let me know of any specific interests or issues that you might have.
- **Ask questions**. For those of you who aren't comfortable asking questions in public, please write your questions down and hand them to me during a break.
- **Share your experiences** with the class. What has and hasn't worked for you? If I say something that doesn't fit your experience, say something!

We look forward to great seminar.

Extreme Programming, Test-Driven Development, and Unit Testing

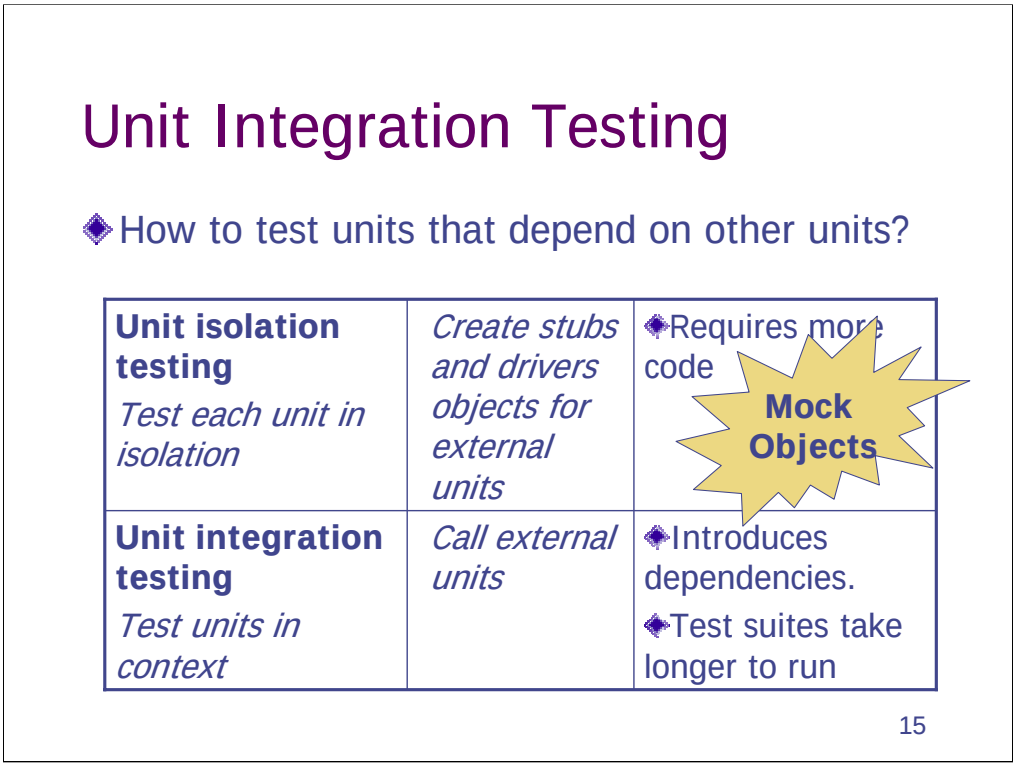


Unit testing has a very specific meaning. It is testing isolated units of code. Typically code often depends on other libraries or components to function correctly. Proper unit testing stubs out these calls so that the units can truly be tested in isolation. This can be laborious and often isn't done.

Sometimes "unit testing" is used to label a partial integration test, where a unit is tested with its underlying components. This is often cheaper to develop, although it may be harder to set up and execute (because of the dependencies).

If your developers have actually developed unit or partial integration tests, learn more about them. What would it take for you to run them? These can often be useful for configuration testing or problem diagnosis.

In some places, "unit testing" simply means "whatever testing is done by the programmers, if any". Usually this amounts to targeted feature testing of their private builds. The quality of this testing is usually quite variable, although most of the stronger programmers do a laudable degree of testing their own code before releasing it to the team.



A unit is really the smallest amount of code that can be tested. The meaning of the term varies some from language to language. In Java and other object-oriented languages, a unit is a class or a group of related classes, called variously a cluster, package or subsystem.

Typically a unit is the work of a single developer or at least can be understood in detail by a single person.

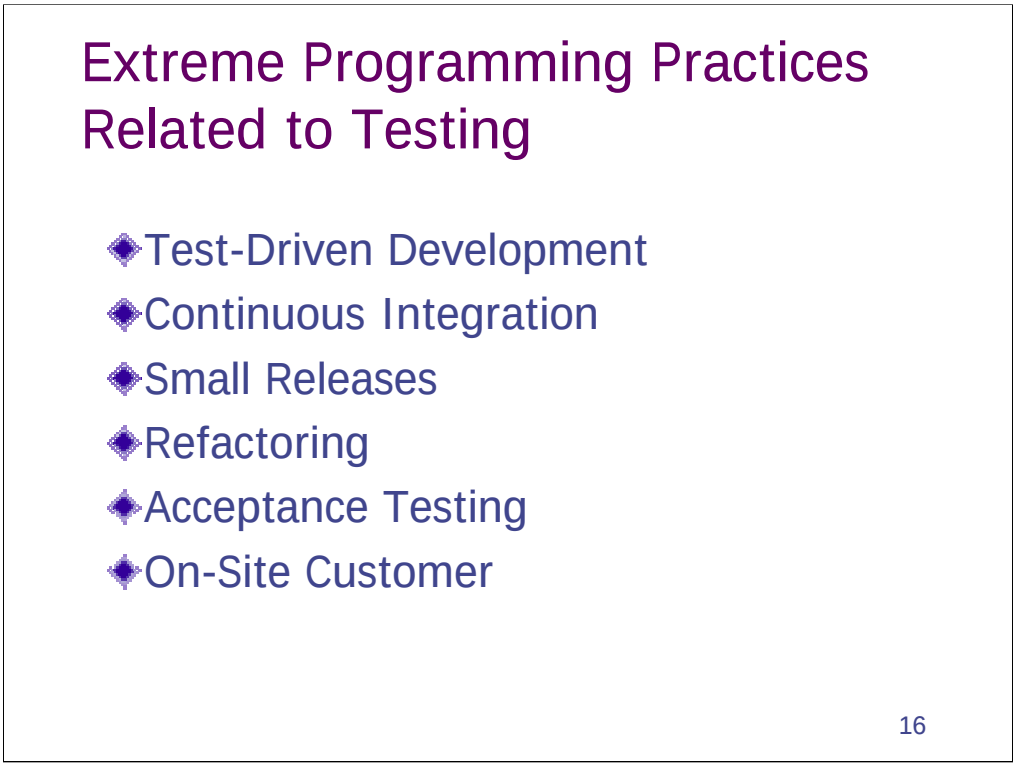
There are really two different paradigms of unit testing. To keep them distinct, I refer to them as unit isolation testing and unit integration testing.

With unit isolation testing, each unit is tested in isolation. In order to be able to test units that would normally call other units, you must create stubs or simulators that pretend to behave like the units that would normally be called. Building this scaffolding can be a significant effort.

With unit integration testing, high-level units can't be tested until the low-level units that they depend on are available. This saves the effort of building stubs, but also requires that the software be tested in a realistic context. For example, if you are testing EJB's that run on an app server, you'll need a test driver that facilitates running tests there. Nonetheless, this seems to be the more popular approach to unit test automation.

I find that programmers who want tests to be automated tend to gravitate towards unit integration testing, rather than system testing of a GUI. This is because this kind of automation takes advantage of the most readily available interfaces.

If you are working in an environment where automation has simply been mandated by senior management, then this really is what I'd suggest pursuing.



Should Testers Go Along with This?

Some say XP is an excuse for hacking and an invitation to poor quality.

I think XP is exciting and will improve the practice of testing in the industry.

- Unit testing is now very popular.
- Open source frameworks for acceptance testing are having a major impact.

Concerns for Testers

Acceptance testing has not been well adhered to by XP teams

Only verifying stories focuses on success rather than risk

Clumsy story definition impacts testing more than development

Need to iterate testing too

How Testers Can Prove Their Value

Demonstrate a helpful perspective in defining software expectations.

Provide information. Don't act like a gatekeeper.

Adapt to changing project goals.

Work with what you have, ask for what you need to know, document what you can.

Extreme Programming Glossary

- Customer
  - A specific role in the XP process. Responsible for defining user stories and setting priorities. The “customer” is the on-site representative of the customer. This person often isn’t actually the customer, which can lead to confusion. Testers often must assume this role.
- User Stories
  - A specific description of a task a user should be able to accomplish using the software.
- Customer Test
  - Sometimes referred to as an *acceptance test* or *functional test*.
  - A test for the system from a user perspective, based on *user stories*. What some people would call a *system test.*
  - Verifies completion of a user story.
- Programmer Test
  - Another name for a unit test.

17

Refactoring
Improving the Design of Existing Code

- Cleaning up code.
- Refactoring doesn't change external behavior, but makes it easier to understand.
- How do you know if the refactoring broke anything?

  Testing!

- *Refactoring*, by Martin Fowler
  - Testing is an integral component to refactoring.
  - 9 of the 17 "sound bites" mention testing.

18

A common problem is code entropy. With time, programmers become more and more reluctant to change a body of code. Often the original programmers have left or moved on to other projects. With time, programmers fear that any change they make may break something. Often this is a sign that the original code wasn't well structured or that later patches were made hastily.

Refactoring is a fancy name for rewriting code. A major goal of TDD and Refactoring is to keep software "soft" – keep it malleable so that programmers will have the courage to continue to make changes to it. A worthy goal.

Test-Driven Development Red-Green-Refactor

1. Write a test, then run it. Make sure it fails.
   RED
2. Make the test pass.
   GREEN
   - Use the simplest design that will work.
   - Bad design (duplication, etc) is OK!
3. Refactor to improve the design.
   REFACTOR
   - Add complexity only when tests demand it.
   - Tests ensure that refactoring didn't break anything.

19

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
  - Allows Refactoring
  - Focuses programmer on how callers will use the code.

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

20

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...

21

New Orleans XP Universe 2003

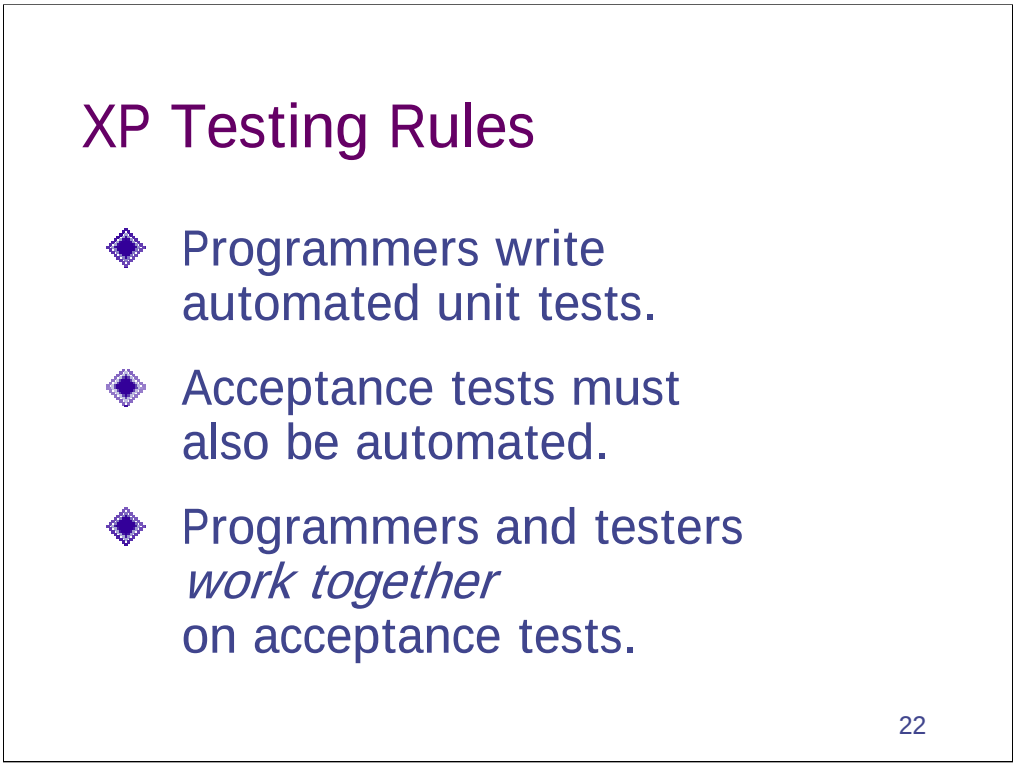
Had a full testing track

Developers eager to talk about testing

Calgary XP Universe 2004

Testing keynote by Brian Marick

Testing discussed in papers and open-space sessions



Over the past couple years, I've been learning a lot about Extreme Programming. I find it to be an interesting approach to software development. I've been particularly interested because of the attention it gives to automated testing, which has been my area of specialty for over a decade.

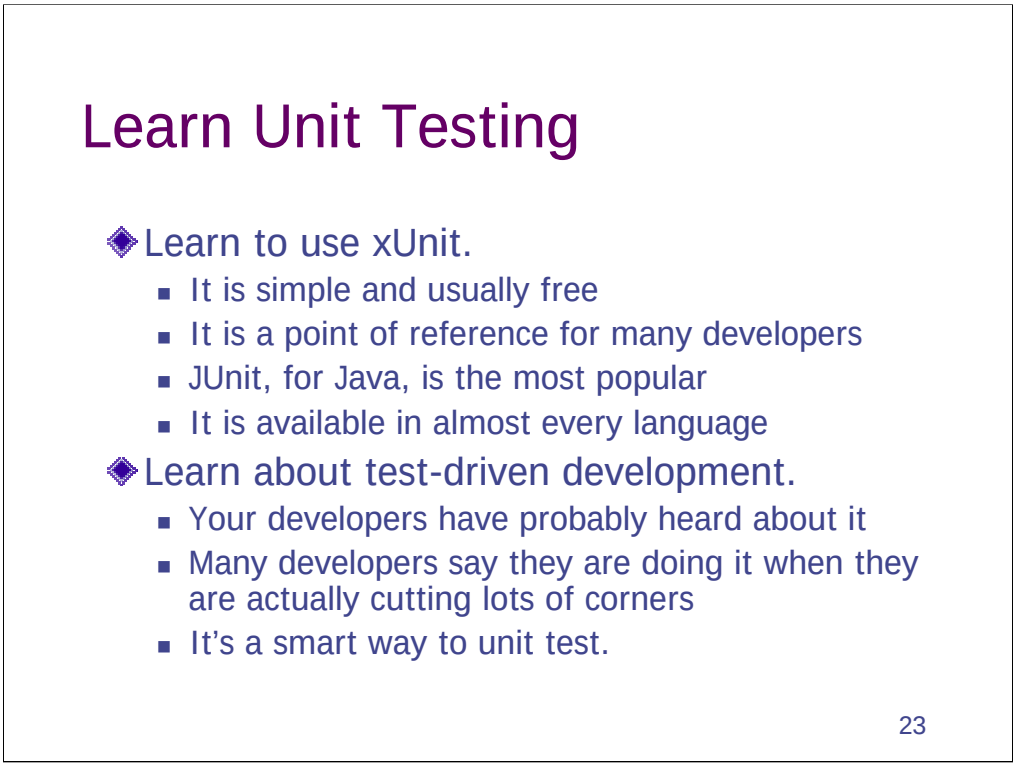
XP has three rules governing automated testing:

> Programmers are required to write unit tests for all their own code.
>
> "Acceptance tests" (system tests, user-perspective tests) are also supposed to be written and automated.
>
> The programmers and the testers are expected to work together to automate the acceptance tests.

In talks in the past I've said that successful test automation requires:

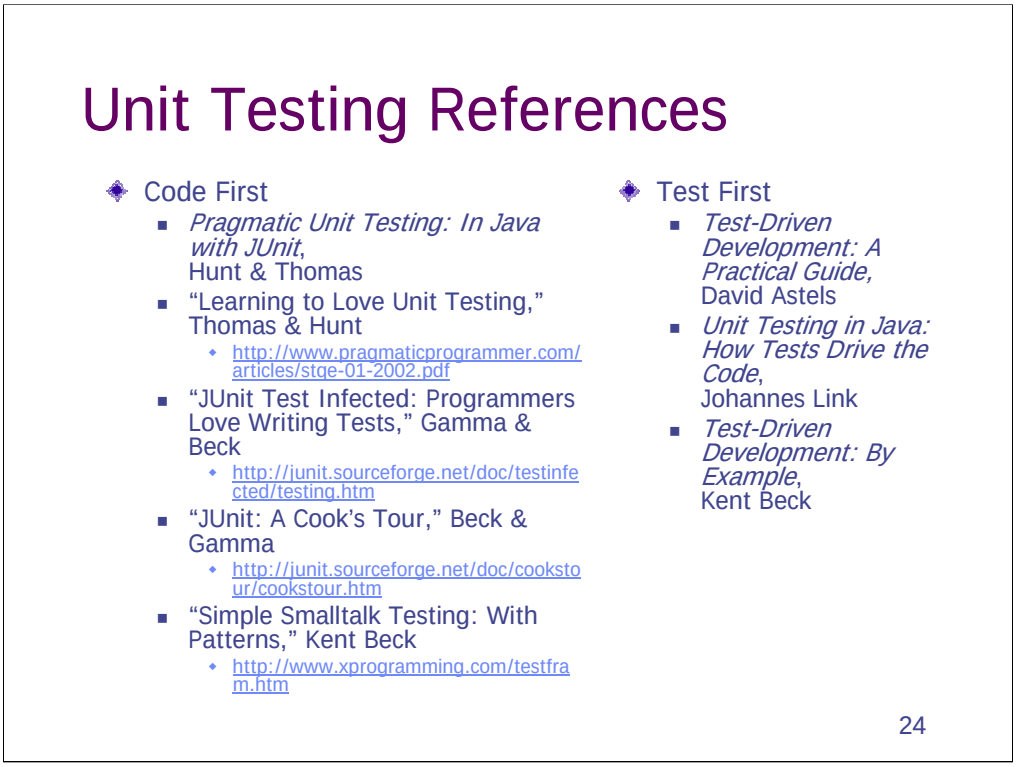
> dedication to automation (rather than treating it as a spare-time activity),
>
> commitment by the entire team (rather than just one or two automators),
>
> and a commitment to automation from the start (rather than trying to automate a manual process later).

This advice was based on my experience with lots of different projects using different methodologies. Now it turns out that XP is saying exactly what I've been saying all along.



In general, I find test-driven development to be the most interesting thinking happening today that relates to test automation. Test-infected developers are solving lots of test automation problems that have been around for years. There are many extremely intelligent, committed and professional programmers in this community.

Many testers are put off by the zealousness of many of these people. They may not have all the answers and they may be overconfident about some of their practices. Nonetheless, this is a community worth learning from. I've learned a lot.



Much of the recent writing around unit testing is in the context of test-driven development. Many people may find it easier to learn about the two topics separately. This is why I have selected several references that do not assume you are using test-driven development, even though I do in fact prefer writing my unit tests first. (I unit test my system testing libraries.)

All of these references cover xUnit. All but one address JUnit, for Java. (The other covers sUnit for SmallTalk, the original xUnit framework.)

Pragmatic Unit Testing is one of the few recent books about Unit Testing that systematically addresses the issue of whether the unit tests are adequate. It's a thin book and covers the basics well. It's my top recommendation.

Beck's articles on his testing framework (including those co-authored with Gamma) may be more accessible to many testers than his book. They are written with a more pragmatic tone.

Astels' and Link's book have excellent coverage of many extensions to JUnit. Link's title is misleading: it also covers acceptance testing of web and GUI interfaces.

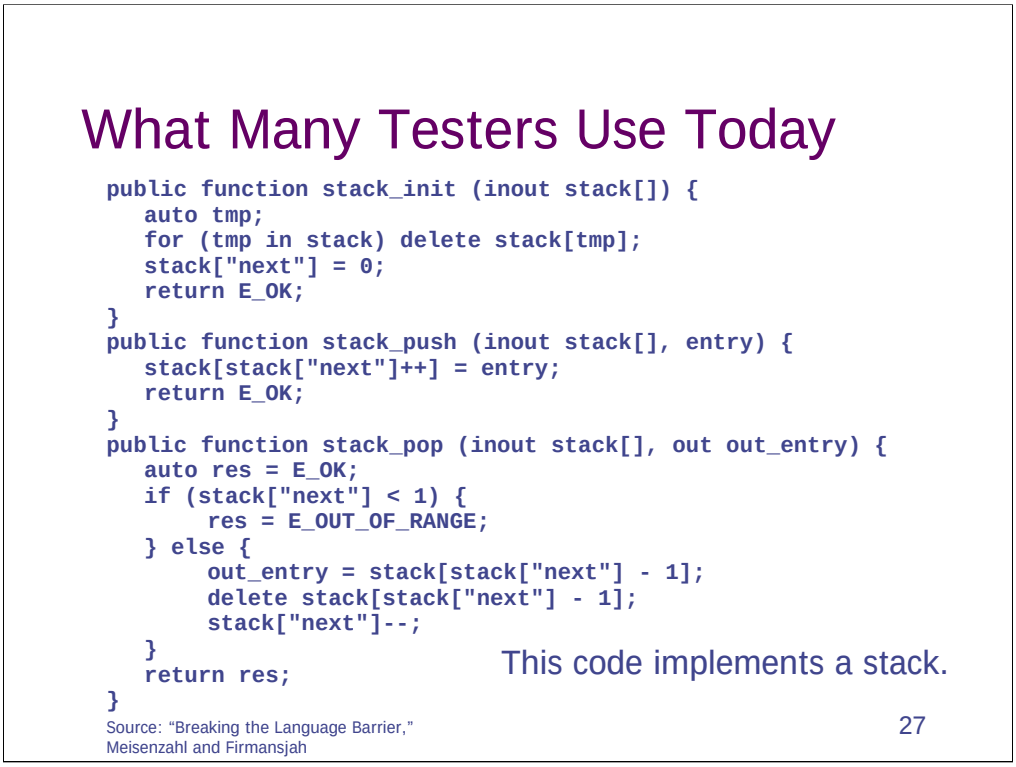
Languages for Testing

- Tie other components together
- Describe tests
- Provide dedicated libraries

YAML is a new data presentation format that is designed to be easy to read.

http://www.yaml.org

http://yaml4r.sourceforge.net/cookbook



The Problem with Vendorscripts

"Vendorscript" is a term I use to describe the proprietary languages that are built into many test tools. I've used a lot of different test tools. They used to all use vendorscripts: even just five years ago. But in the past few years some vendors have started to package standardized scripting languages with their tools instead. I think this is a great trend that I really want to encourage.

I wrote a column for Stickyminds.com stating my views on this matter. I'd felt strongly about this matter for some time, but what really prompted me to speak out was an article in a trade magazine. The authors of the article described how they worked together to add stacks and queues to the vendorscript of their test tools. The code they wrote is contained on this slide.

The theme of the article was that team work would help testers achieve great things. But the lesson that I drew was that testers had to jump through extra hoops to do basic things when they were saddled with crummy languages.

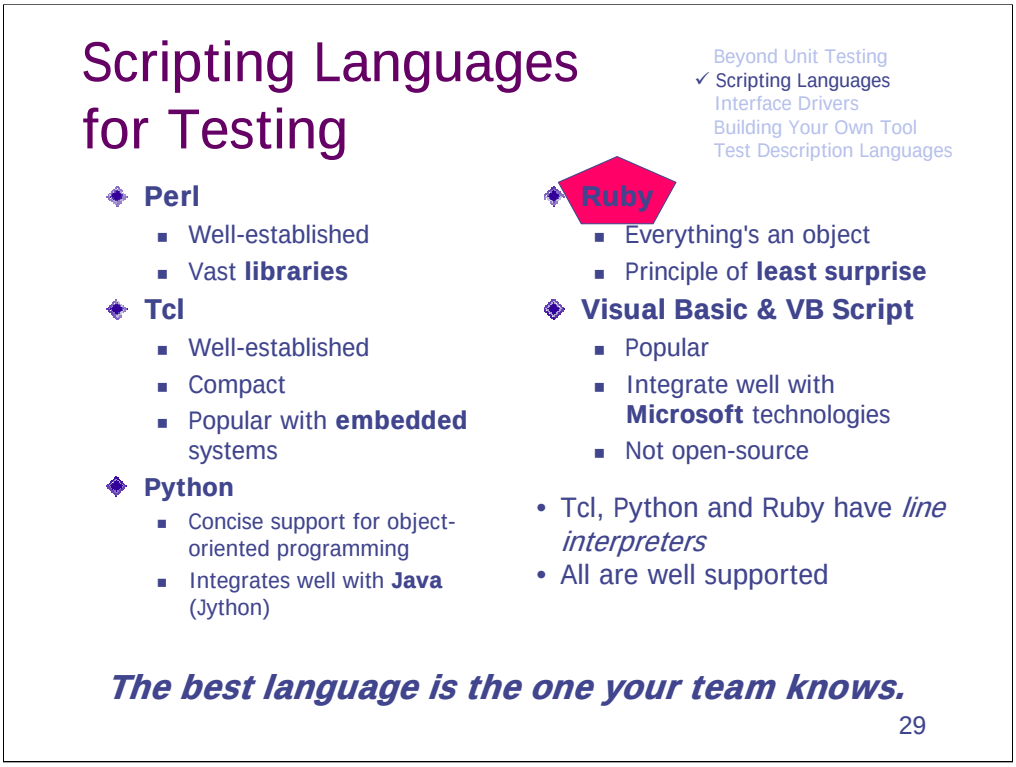
They were using a tool that used a C-like vendorscript. Now stacks and queues are basic data structures that are covered in introductory computer science classes. Implementing them in C is not complicated and does not require team work. But the authors of the article weren't using C. They were using a "C-like" vendorscript.

The Problem with Vendorscripts (contd.)

So what is a “C-like” vendorscript? It is a language that has the same syntax as C, but lacks support for using pointers. If you know C, you know that you use pointers for everything. You use them for stacks and queues, for example. But without pointers, you face a challenge. Now it takes teamwork and cleverness. The achievement is sufficiently momentous that these two authors chose to use it as the basis for an article on teamwork.

Writing C without pointers is like writing English without the letter E. It can be done. And it’s not too hard to read it, although it may sound somewhat stilted. But writing English with E’s is a real challenge.

Now I don’t want to make light of their achievement. But I do want to suggest that the lesson to be drawn is not that testers need to combine forces to overcome the problems of test automation. Rather, testers deserve substantial languages that will help them focus on real problems. They need to stop using vendorscripts, and vendors need to stop giving them to them.



These are the scripting languages that people are using – and like using – for testing. I've had a chance to use all of them.

Languages not listed here are JavaScript and Rexx, mainly because there doesn't seem to be much activity around them. The Rexx community (IBM) seems to be moving to Python.

All of these languages listed here, except for VB and VB Script, are open-source.

Another up and coming scripting language is **Groovy**. Like Jython, it runs inside the Java Virtual Machine.

A useful essay on why Python is easier to use than Perl.

http://www.garshol.priv.no/download/text/perl.html

Scripting Languages

	Free	Mature & Libraries	Line Inter-preter	Regexp	Tem-plating	Java Embed	COM	OO
Ruby	✓	*	✓	✓	✓	?	✓	***
Python	✓	**	✓	✓		✓	✓	**
Perl	✓	***		✓	✓		✓	*
TCL	✓	***/*	✓	✓	✓	?	?	*
VBScript & WSH	✓	***		✓			✓	**

30

Perl has the most libraries of any of the languages listed here. For many people, this is a good reason to use Perl.

A line interpreter allows you to type a line of code and then see what it does. I find this to be the very best tool for learning a scripting language. The lack of a line interpreter for Perl is to me it's greatest weakness. Introductory texts for the languages that have them, use them to demonstrate how the language works. (Perl zealots often claim that "perl –d" provides an interpreter. It's never worked for me, nor have I seen it mentioned in any introductory Perl books.)

Regular expressions use a powerful syntax for pattern matching. They are critical for effective verification. Sadly, most vendorscripts lack them.

Templating is the ability to interpolate variables in strings. For example in Ruby:

```
print "name: #a.name value: #a.value"
```

This is often very handy in testing. I think the lack of native support for general templating is Python's greatest weakness.

Python, however, has a version that runs inside a JVM (Java). This often makes it the best choice for testing java applications.

COM support allows a language to call COM libraries. This is often important on Microsoft platforms.

Some languages support object-oriented (OO) programming better than others. This is often not that important to testers, but can be a big deal for programmers. The more code you write, the more you may find yourself wanting to use OO features of the language. It's not needed for short scripts.

Ruby Code

```
# pause the job
pause_or_stop_form =
   get_form_by_action("pause_or_stop_job")
pause_or_stop_form.elements("pause").click
@iec.wait

# restart the job
for form in @iec.document.forms
  if form.action == "start"
    paused_form = IEDomFormWrapper.new(form)
  end
end
get_element_by_value(paused_form, "foreground").click
@iec.wait
```

31

This is a snippet of Ruby code to give you a sense of it's syntax. Ruby uses "end" to mark the end of "for", "if" and similar statements.

Learning Ruby

- "Programming in Ruby," Thomas & Hunt
  - http://www.pragmaticprogrammer.com/articles/ddj_ruby.pdf
- *Programming Ruby*, Thomas & Hunt (Pickaxe Book)
  - http://www.pragmaticprogrammer.com/ruby/
- *Why's (Poignant) Guide to Ruby*, Why The Lucky Stiff
  - http://poignantguide.net/ruby/
- *Ruby User's Guide*, Matz, et al
  - http://www.rubyist.net/~slagell/ruby/

32

The Thomas & Hunt article first appeared in Dr. Dobbs. It gives a good overview to the language (free).

The Pickaxe Book is the most popular book on Ruby. It assumes you already know Java or C++. It is a good introduction to the language and it also has a very good reference section. An electronic version is packaged with Ruby for Windows (free). It is also available on the web (free).

Why's Guide is a work in progress on the web (free). It is written for the non-programmer. It is so funny that some people have learned Ruby even though they didn't intend to.

Ruby User's Guide is another free online book. This is an English translation of a book writing by Matz, Ruby's creator.

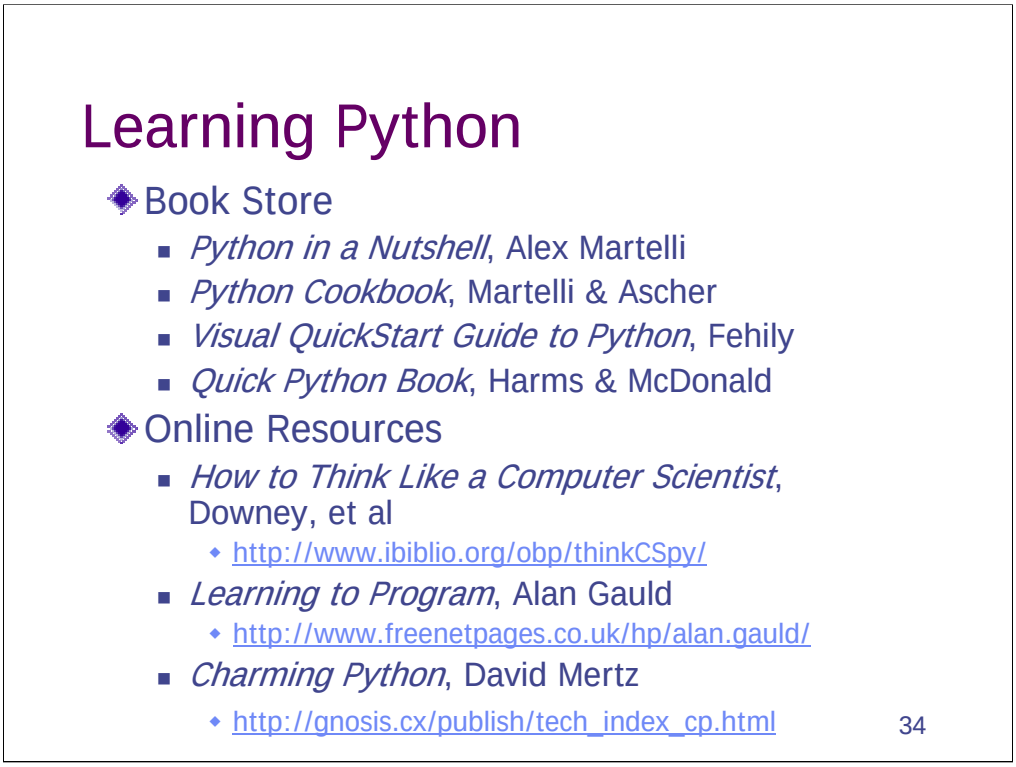
Python Code

```
def test_copy(self):
   "test copying to the same machine."
   InputFile = r"C:\Input\text\busy"
   OutputFile = r"C:\Output\text\test-j.txt"

   # Connect to node
   machine = self.machine
   machine.connect_by_name()
   assert machine.cd_node.getConnectionInfo().getNodeName()
   == machine.name

   # Remove output file from prior execution
   if os.path.exists (OutputFile):
       os.remove (OutputFile)
```

33

This is a snippet of Python code. Python uses indentation to mark the contents of blocks for “def”, “if” and similar statements.

A lot of people get confused about Python vs. Jython. When I say “Python” I mean both. This code is actually Jython code. You can see some calls to Java objects in it.



I learned Python last year, trained several testers to use it, and then lead the development of a Python-based test suite for a client. Python is a great language for testing. Mostly, i learned it from books, reading eight different books in the process. My favorite is Python in a Nutshell by Alex Martelli (O'Reilly). I found it well-organized and with extremely accurate and detailed descriptions of the inner workings. Mostly you don't need to know this kind of thing, but when you do, you have it. Unlike most Python books, this gives equal treatment to the two slightly different versions: the one written in C ("CPython") and the one written in Java ("Jython"). Another excellent book is the Python Cookbook by Martelli and Ascher (O'Reilly). It contains lots of meaty Python examples. Some are useful because they illustrate how to use certain features or libraries. Others can be reused in whole to solve problems you have. I did.

The Visual QuickStart Guide to Python by Chris Fehily (Peachpit) is an excellent book for helping anyone understand the basic syntax of the language. I think it would help any one who wanted to learn Python. It uses an easy to read split-page format with code examples (often interactions with the interpreter) on one side and explanation on the other. Some users may find that it is the only book they need.

I learned a lot from The Quick Python Book by Harms and McDonald (Manning). It has a nice introduction to the language that will be helpful to people who already know a couple other languages. It also has a useful reference section, although after learning the basics of the language, i found myself depending on Martelli's Nutshell book instead.

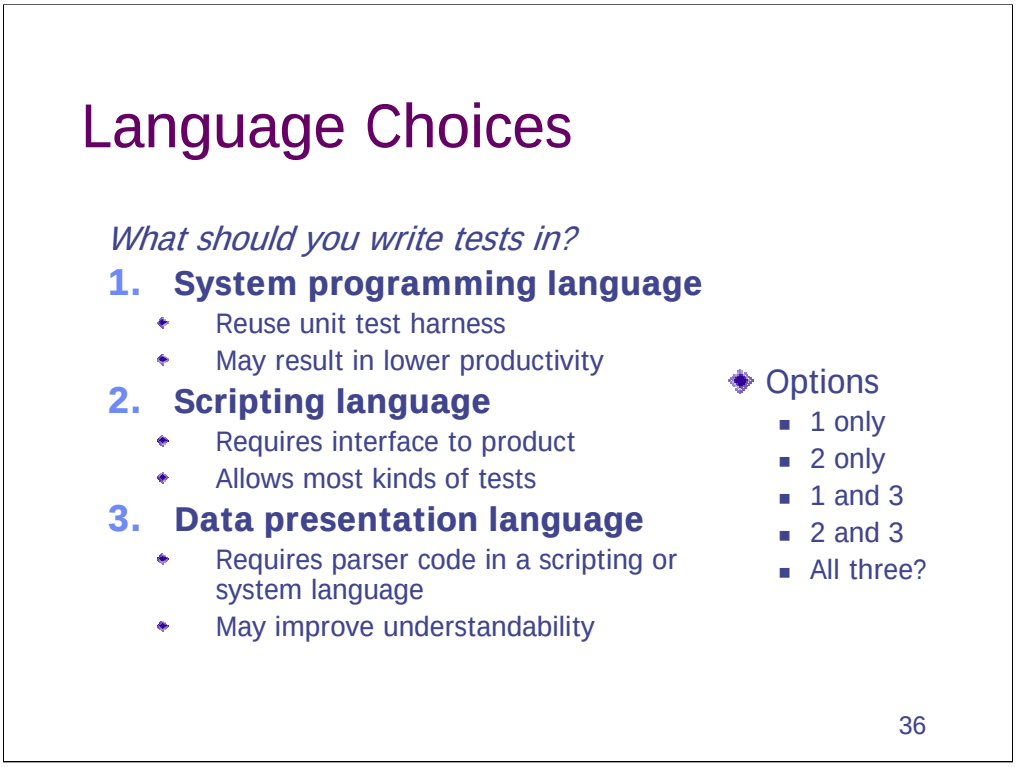
There are also two online books that teach Python to people who don't know how to program at all. I haven't read them, but if you are a manual tester looking to get into automation, one of these might just be the thing. You can read them online for free or purchase a bound version. They are How to Think like a Computer Scientist and Learning to Program.

Python also comes with very good reference material bundled with the language itself.

Ruby vs. Python

- Why I recommended Python to a client
  - Supported on more platforms
    - Especially OS390
  - Able to call Java API's (Jython)
  - More intro-level books
    - Therefore easier to learn
  - Packaged IDEs
- Where Ruby is better
  - Built in "templating"
    - `"#{ruby_code}"`
  - Easier to write object-oriented code
  - Better support for functional programming

35



Data presentations languages require a parser, which must be in a system or scripting language. We'll discuss data presentation languages further when we cover test parsers later in the notes.

Use Scripting Languages for Testing

I strongly believe that standard scripting languages are what should be used for most system testing. They are powerful, easy-to-learn and support interactive learning. I mostly use Ruby for personal projects, including the classes i teach, but i am not pushing Ruby as the right solution for everyone.

In particular, I'd also like to see more testers using Python. Since i wrote my note about why I'm using Ruby, I've had a chance to select Python for a test team and train several of the staff in the language. I was surprised at how successful that was. I selected Python over Ruby for two reasons. First was that Python is available on a lot more platforms. These testers had to work with software on many platforms: Windows and Unix, as well as Tandem (know called HP NonStop), AS400 and Mainframes. Python is available on all of these platforms. Even more important, there is a version of Python that can run in a JVM. (This version is called Jython.) This turned out to be key, as it allowed us to directly call Java API's from Python, and opened up a whole new approach to testing.

The second advantage that Python has over Ruby is that it is just a lot more mature. There are more books available, including introductory books that don't presume you already know a couple other languages. I reviewed these recently, and only included the best that I'd found. It was nice to have had such a wide selection.

A number of testers have had significant success using Perl. It's even more mature than Python and available on more platforms. It has more libraries and nifty utilities, such as one that can compile Perl code into an executable (which can then be run on machines that don't have Perl installed). And if you already know Perl or are in a shop with a lot of Perl programmers, then i encourage you to seriously consider using Perl for your testing.

But I hesitate to recommend Perl to others. I've mentioned before that it's syntax leaves something to be desired. I do have some sympathy for Perl's tangled syntax. It was largely designed as a replacement for Unix shell scripting with utilities such as grep and awk and sed. And its syntax is significantly less confusing than what it replaced. But I'm coming to realize that Perl's main shortcoming is in its lack of an interactive interpreter. With the Unix shell you had this. And you have it with Python and Ruby. On a couple occasions, I have heard Perl defenders respond by saying that you can get an Perl interpreter if you type "perl -d -e 42". I'm sure this works for them, but i have tried this on several machines, with several different versions of Perl, and its always given me errors. Moreover, I've never seen this advice given in introductory Perl materials. So I'm going to stick to my position that Perl does not have an effective interpreter.

Use Scripting Languages for Testing (contd.)

A line interpreter allows you to type one line of code and see what it does. It is a great way to learn a language. Unlike me, there are lots of good testers who are not multilingual. They would rather spend their time understanding the software they are testing as a whole, as well as the needs and expectations of its users. So if they are going to learn a language, it has to be easy. Line interpreters make language learning easy.

In the early Eighties, I taught Basic and Logo to both kids and adults. Both languages have line interpreters. I taught simply by giving the students stuff to type in, and then encouraged them to develop rules to explain the behavior they were seeing. In effect, i encouraged them to converse with the computer language. Learning is accelerated with the shortened the time between writing code to seeing its effects. See if you can understand this Ruby interaction:

```
>> a = [1, 2, 3]
=> [1, 2, 3]
>> a.last
=> 3
>> a.each {|x| p x * 2}
2
4
6
=> [1, 2, 3]
```

These kinds of line interpreters facilitate exploration. You start by exploring the language, then its libraries, then software under test that the libraries drive. I don’t really understand how any one who really likes exploratory testing could be satisfied with a language like Perl.

Another popular scripting language for testing is Tcl. Tcl has a line interpreter (called tclsh) like Python and Ruby. And it has a simple syntax. It really has two things going for it. First is Expect, a tool built atop Tcl. Expect is a great tool for automating Unix-style command line interfaces. Second is that, being a simple language, Tcl itself is smaller than Perl or Ruby or Python: this makes very suitable for testing networking equipment and other embedded systems that may not have enough storage to support a large language. I had an opportunity to work with a large Expect-based test suite years ago. It tested Unix system administration software.

Use Scripting Languages for Testing (contd.)

VBScript is Microsoft's scripting language. It is a simplified version of Visual Basic that works with Windows Scripting Host (WSH) to provide scripting ability on Windows operating systems. It could be used for testing, but I'm not aware of it being used for more than very simple tasks. Testers building significant test suites seem more likely to use Visual Basic itself. One reason may be that VBScript lacks a line interpreter. And Visual Basic comes with a much richer IDE than what you get with VBScript. VB Expert Mary Sweeney says that she writes VBScript by first writing VB in its IDE and then, when its working, stripping out the extra stuff (mainly the variable declarations) to turn it into VB. I worked on a project last year for a Microsoft shop. They already had some tests written in VB. I extended this suite and encouraged them to use it for a wider variety of tests. I looked at VBScript at the time, but really couldn't find a reason to favor it. They hadn't moved to dot-Net yet. With dot-Net, I'm not really sure what the most agile testing language will turn out to be.

So my list of scripting languages for testing is: Ruby, Python, Perl, Tcl and VB/VBScript. These are all general purpose languages, with large user communities and continuing support. There are other scripting languages, but these are the ones that seem to be used for testing. There are probably some mainframe testers using Rexx, but even there, I'd think they'd be better off using Perl or Python (if they can convince the sysop to install them).

There are three other types of languages that are sometimes suggested as good testing languages: vendor-specific tool languages, general programming languages, and data specification languages. Ultimately, I find each of these insufficient for testing.

Interface Drivers

- Interact with product software under test

Types of Interface Drivers

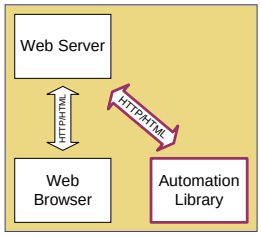
- Web Protocol Drivers
  - Automate protocol interaction. E.g. HTTP
- Web Browser Drivers
  - Automate the operation of a web browser
- Java GUI Drivers
  - Automate Java GUI interfaces
- Windows GUI Drivers
  - Automate Windows interfaces. E.g. Win32, MFC
- Character Interface Drivers
  - Automate a command-line interface

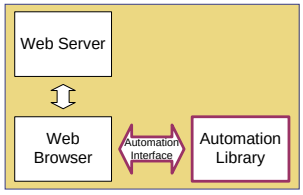
41

Getting a suitable interface driver for your product is often the most critical aspect of homebrew automation. Commercial tools typically package drivers for lots of different interface technologies. But you rarely need this kind of diversity to test a single product. You'll need to find a best fit for the kind of interfaces that your product has.

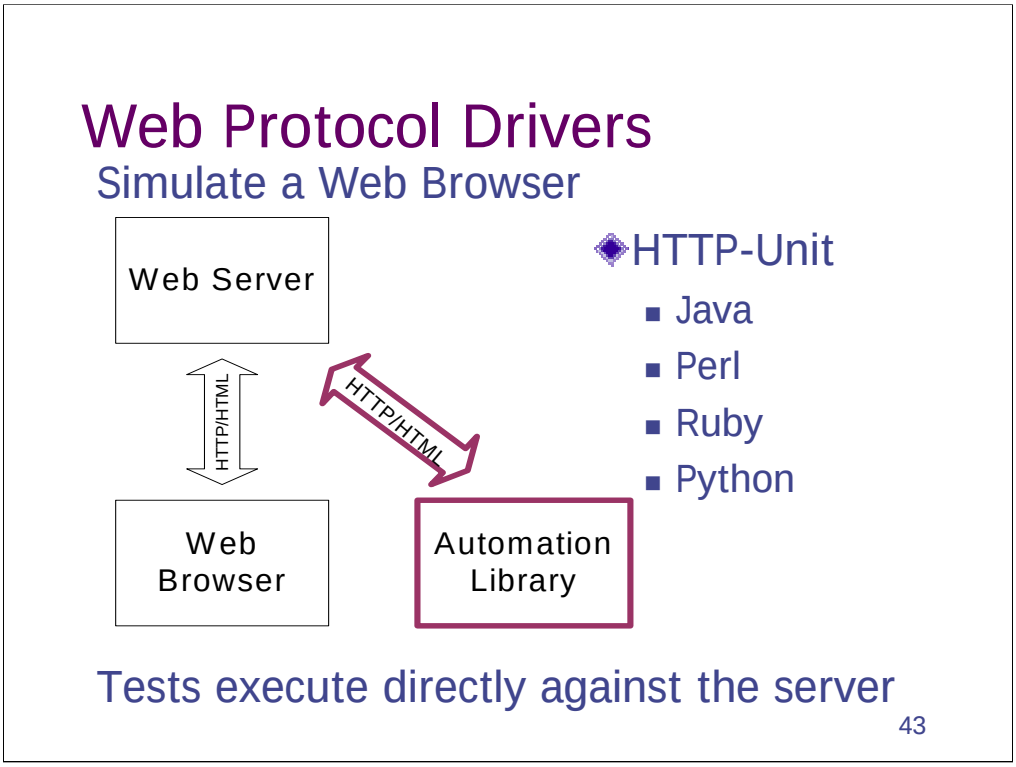
Web Interface Drivers

- How do your tests access the product server?
  - Simulation (Protocol Drivers)
    - Access the server in the same way as a browser would.
  - Automation (Browser Drivers)
    - Drive the browser using automation interfaces.



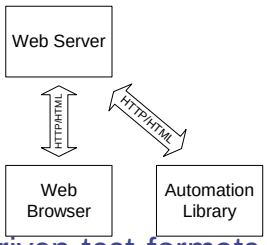


42



This is the most common approach, and there are many extensions to the base tools in this category.

Web Protocol Drivers



- A very popular category
- Some use parsers to support data-driven test formats
  - XML, HTML
- Some support embedded scripting languages
  - Namely embedding Jython in Java
- Focus varies between functional and load testing
  - Functional tools tend to offer better browser simulation
  - Load tools tend to offer better support for concurrent testing
  - But most can do some of either
- Some support many protocols
  - Including ones not supported by browsers
  - E.g. SOAP

44

Web Protocol Drivers for Functional Testing

	Tool	Tests
HttpUnit, popular http://www.httpunit.org/	Java	Java
jWebUnit, extends HttpUnit and Fit, http://jwebunit.sourceforge.net	Java	Java & HTML
Canoo WebTest, extends HttpUnit http://webtest.canoo.com	Java	XML
HtmlUnit, similar to HttpUnit http://htmlunit.sourceforge.net/	Java	Java
libwww-perl, general tool, e.g. spiders http://ftp.ics.uci.edu/pub/websoft/libwww-perl/	Perl	Perl
WebUnit, based on HttpUnit http://www.xpenguin.biz/download/webunit/index-en.html	Ruby	Ruby
Puffin http://www.puffinhome.org/	Python	XML
WebInject http://www.webinject.org/index.html	Perl	XML

45

More Tools:

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

ITP

Another java-based tool supporting XML-based test scripts.

http://www.incanica.com/itp.html

Jameleon

An XML-based test case front-end for tests using HttpUnit or jWebUnit

http://jameleon.sourceforge.net/

Curl

cURL is a different kind of tool. It is a command-line tool for work with URLs. Some testers report using it for simple testing tasks, such as link checking.

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

46

Open source Java API for interacting with web servers

- Starts a session
- Sends requests
- Receives responses
- Navigates links

API makes it easy to write assertions to validate web pages

- Tables, forms, links, headers, response codes, etc.

Response is also available as a DOM document!

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>
```

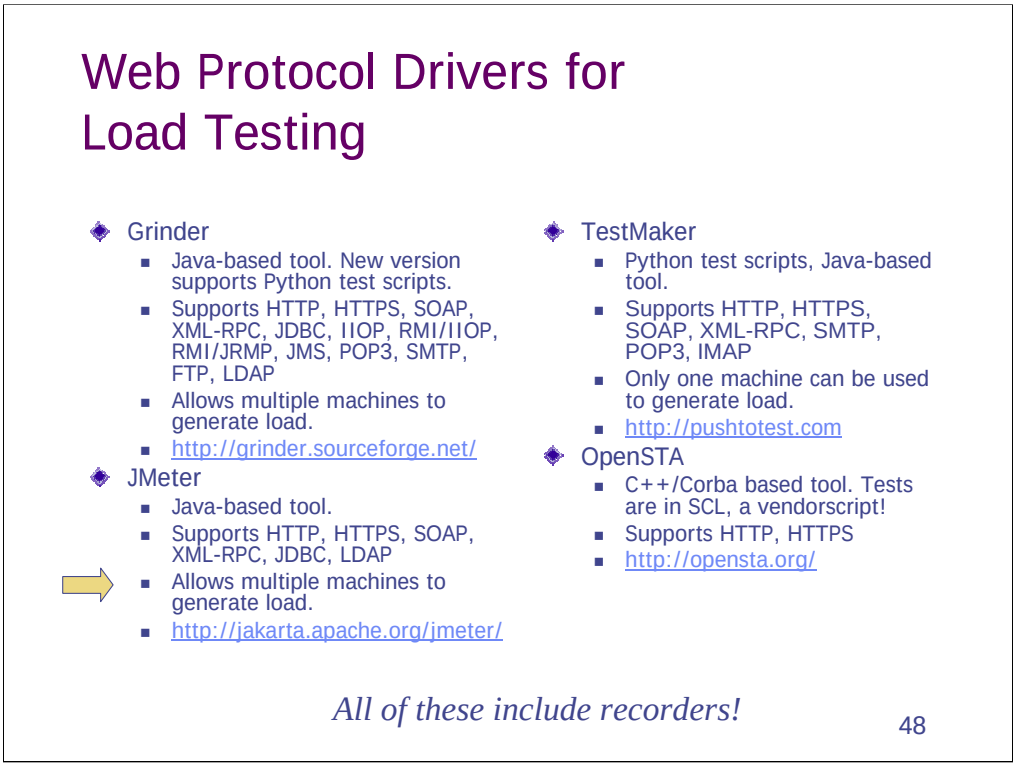
Example Courtesy of Mike Clark

47

Canoo WebTestTest steps are expressed in XML and run as Ant tasks

Under the hood, test steps are executed using HttpUnit

XML tools can be used to write and maintain tests



TestMaker has a proxy service that allows it to record the communication between a web browser and a server. It spits this out as a Python script. This is typically used for creating load tests, but it really could be used for any kind of protocol-based testing. TestMaker itself is written in Java. It comes with support for HTTP, SOAP, SSL and several other protocols. And it's designed to be extensible to easily support other protocols. Frank Cohen, the lead developer, has published a nice book describing how to use the tool.

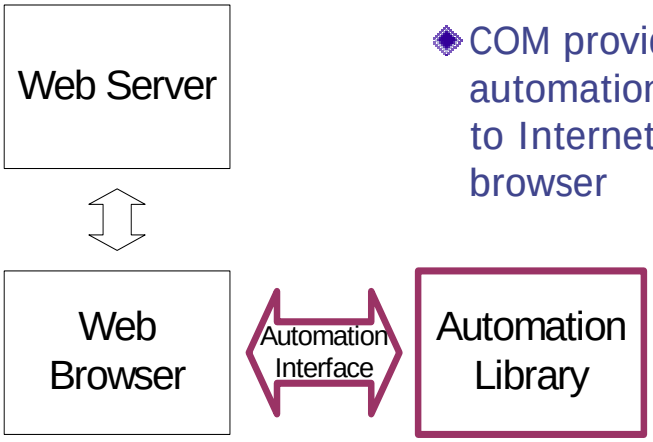
An Overview of Load Test Tools, by Buret & Droze

Excellent comparison of these and other load testing tools.

http://clif.objectweb.org/load_tools_overview.pdf

Web Browser Driver

Automates a Web Browser



- COM provides automation interfaces to Internet Explorer browser

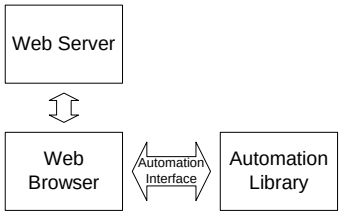
Tests execute against a browser

49

Web Browser Drivers

- **WTR**, Chris Morris et al
  - IE automation in Ruby
  - http://rubyforge.org/projects/wtr/
  - http://www.clabs.org/wtr/
- **Samie**, Henry Wasserman
  - IE Browser automation in Perl
  - http://samie.sourceforge.net/

WTR/IEC Example



50

There is also a recently announced plan to port Samie to Python (Pamie).

Java GUI Drivers

- Several well-developed open-source tools
- Some are more for unit testing GUI components. Others are for system or acceptance testing.
- Use various methods for triggering controls:
  - Use **Robot** to generate native OS events
  - Generate **AWT** events
  - Drive control **components** directly

51

Java GUI Drivers

	Scripts	Recorder	Unit Test	System Test	Event Mechanism		
					Robot	AWT	Compo-nent
Jemmy, Shura Iline Integrated with NetBeans http://jemmy.netbeans.org/	Java		✓		✓	✓ Default	
Abbot, Timothy Wall http://abbot.sourceforge.net/	XML	✓	✓	✓	✓ Default	✓	
Marathon, Thoughtworks http://marathonman.sourceforge.net/	Python	✓		✓		✓	✓
JfcUnit http://jfcunit.sourceforge.net/	Java		✓		✓	✓ Default	
Pounder, Matthew Pekar http://pounder.sourceforge.net/	?	✓		✓	✓		

52

A excellent comparison of these tools is “Java GUI Testing Tools” by Timothy Wall

http://tejasconsulting.com/open-testware/contrib/JavaGUITesting.pdf

An interesting review:

http://www.artima.com/forums/flat.jsp?forum=121&thread=8540

More Tools

http://groups.yahoo.com/java-gui-testing

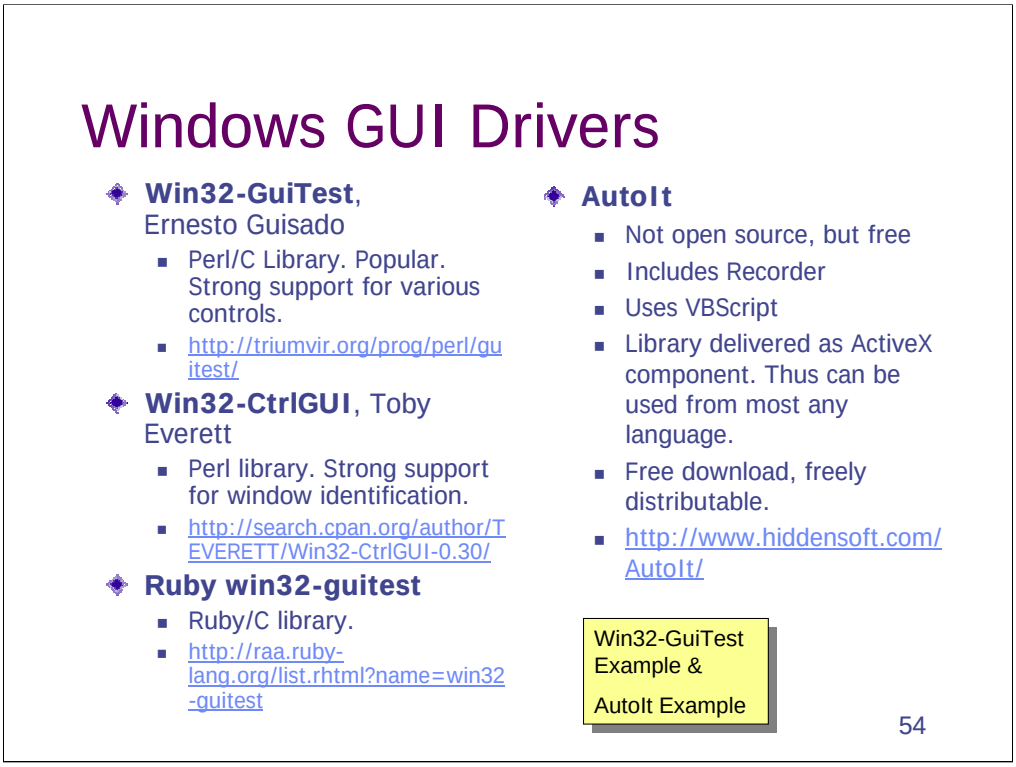
http://www.junit.org/news/extension/gui/index.htm

http://www.superlinksoftware.com/cgi-bin/jugwiki.pl?TestingGUIs

Test-Driven Development (David Astels) provides detailed examples for using Jemmy.

Windows GUI Drivers

- Visual Basic's SendKeys() is common.
- Perl's Win32-GuiTest uses the same approach.
- You may be able to use OLE Automation instead.



Novell AppTester

API typically called from C++.

Distributed as binary as part of Novell's system testing tools.

Not open source.

http://developer.novell.com/ndk/softtestv3.htm

The following is an interesting study in how to create a test tool. It's a good discussion and all the code is included. It ends up really being a study in how not to create a test tool. He relies on an analog approach that ends up (as I'd expect) being unreliable. He tries some kludges, but nothing quite works. Not open source.

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.

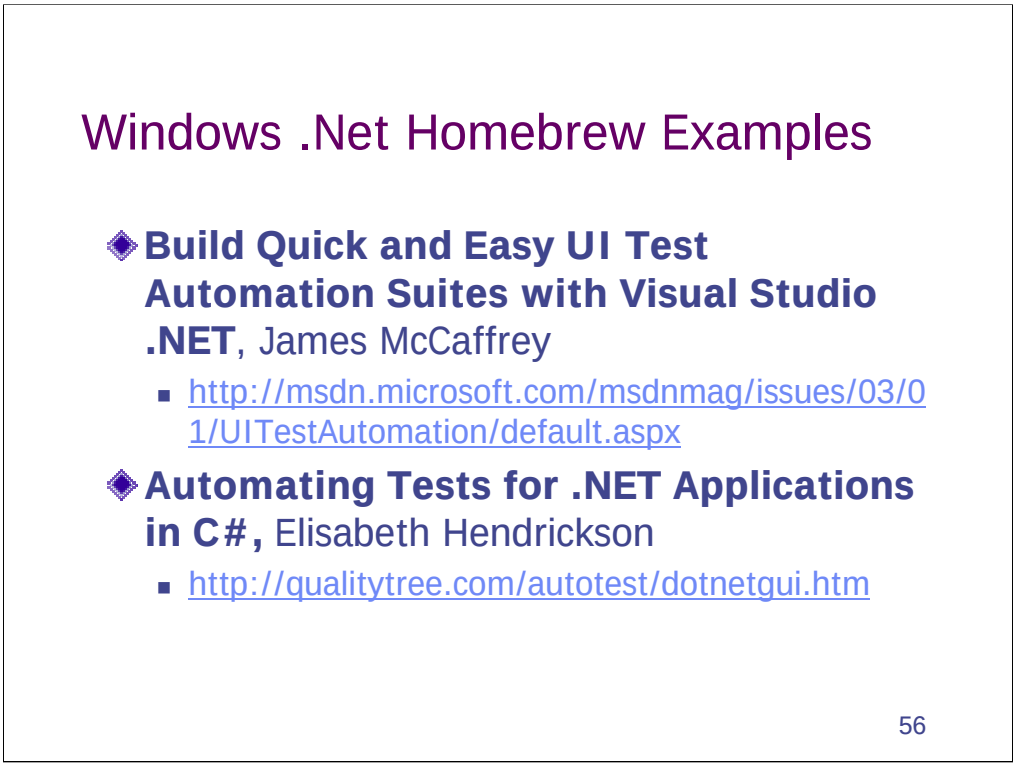
http://msdn.microsoft.com/msdnmag/issues/02/03/bugslayer/default.aspx

Macroexpress is another tool for automating GUI interfaces. It is a cheap shareware tool.

Windows GUI Drivers

- Visual Test
  - A commercial tool that nonetheless has often been used as the core of homebrew solutions.
  - Although packaged as a complete tool (with a Basic-dialect vendorscript) its GUI driver (VTEST60.DLL) can be called directly from VB or C++.
  - Developed at Microsoft, sold to Rational, acquired by IBM. The irony!
  - Sadly, is not being updated for .Net. Doesn't even support Windows XP.
  - http://www.visualtest.com/tools/vt/index.shtml

55



.Net opens many opportunities for homebrew automation. It's built atop COM and OLE, thus supporting those approaches to automation. And it adds reflection, a key technology that has really helped a lot of the Java homebrew efforts.

I don't know of any open source tools, per se. But these are published examples of homebrew automation using .Net.

The Character Interface Driver: Expect

- **Expect**, Don Libes
  - Automates Unix terminals
  - Uses Tcl or [incr Tcl]
  - Long-established.
  - *Exploring Expect*, Don Libes
  - http://expect.nist.gov/
  - http://wiki.tcl.tk/expect
  - http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/expect/
- **Pexpect**,
  - Expect for Python
  - http://pexpect.sourceforge.net/
- **Expect.pm**
  - Expect for Perl
  - http://sourceforge.net/projects/expectperl/

57

From *Exploring Expect:*

3Com does software quality assurance with Expect. Silicon Graphics uses it to do network measurements such as echo response time using telnet. The World Bank uses it to automate file transfers and updates. IBM uses it as part of a tape backup production environment. HP uses it to automate queries to multiple heterogeneous commercial online databases. Sun uses it to sweep across their in-house network testing computer security. Martin Marietta uses it to control and extract usage statistics from network routers. Tektronix uses it to test software simulations of test instruments. The National Cancer Institute uses it to administer accounts across multiple platforms. Cisco uses it for network control and testing. Xerox uses it for researcher collaboration. Motorola uses it to control and backup multiple commercial databases. Data General uses it for software quality engineering. The Baha'i World Centre uses it to automate and coordinate data collection and storage from different telephone exchange locations. Amdahl uses it to automatically retrieve stock prices. CenterLine Software uses it for software quality assurance. Encore Computer uses it to simulate 500 users logging into one system at the same time. AT&T uses it to copy files between internal hosts through a firewall to and from external Internet hosts. Sandia National Laboratories uses it to control unreliable processes that need to be watched constantly. Schlumberger uses it to manage a network environment including routers, repeaters, bridges, and terminal servers all from different manufacturers. ComputerVision built an automated testbed using it.

Tcl Code

```
set saveDir [ChangeRunDir "/"]
set storeDev [lindex $gDevicePath 0]
set restoreDev [lindex $gDevicePath 1]

if { ![info exists sysSingleStoreListing] ||
![FileExistsAndContainsData $sysSingleStoreListing] }
{
   set sysSingleStoreListing [GetTempFile "stoList"
{STATIC} ]
   set storeCmd "select * filesystem single \
        excluding (./vobs, ./vob, ./view, ./vgbig) \
        store to $storeDev compress 1 display
progress 5"

} else {
   set storeCmd ""
   set options "{SKIP_STORE $sysSingleStoreListing}"
}
```

58

This is a snippet from a test suite. This code is Tcl. Eventually the functions called here make calls to Expect, but you don't see the expect code here. On the flip side, this code is harnessed by DejaGnu.

Notes from Pete TerMaat on using [incr Tcl]:

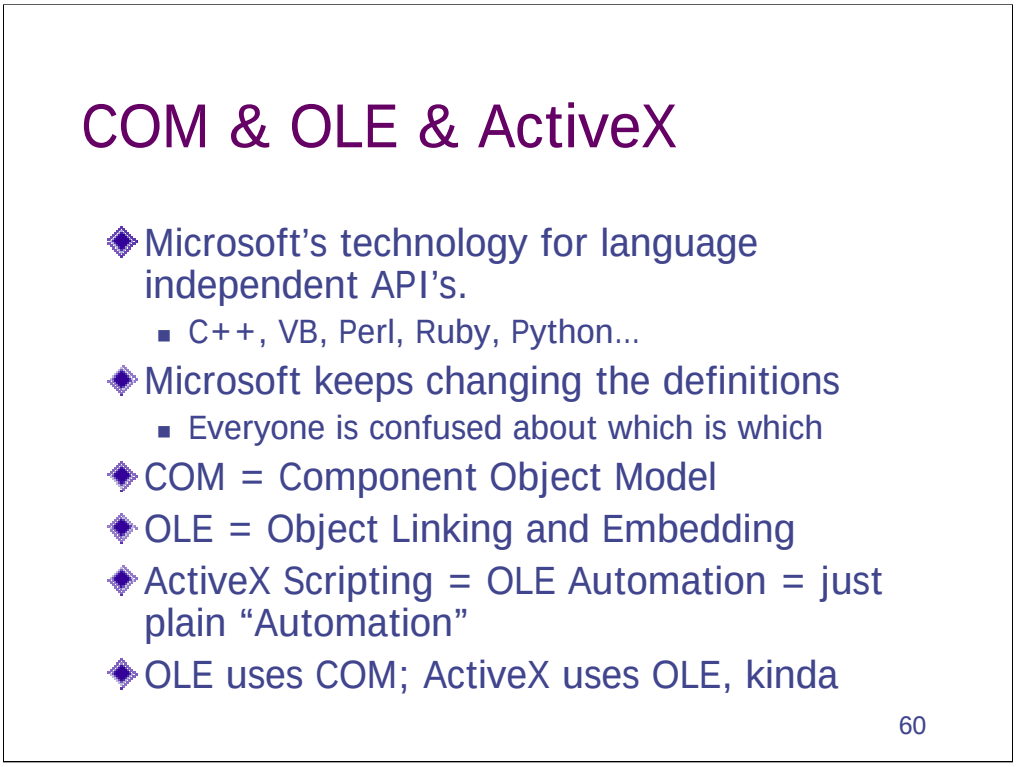
- Network test equipment often has Tcl interfaces.
- Complex data structures can become unwieldly.
- Support for "new" things like XML, SOAP, and whatnot is often slower in coming than with Perl, Python or Ruby.
- Relatively few books.
- Smaller development community than Perl or Python.

Do You Need an Interface Driver?

- Some of the most effective test automation systems drive product software using programming interfaces such as RPC, CORBA, COM, or RMI.
- Most scripting languages support calls to these programming interface technologies.
- No special interface driver is required.
- Use Python (Jython) to drive Java interfaces.

59

Iron Python is a port of Python to the .Net CLR. It could be used to drive .Net interfaces.



This technology was originally developed to facilitate the integration of Word and Excel and PowerPoint into Office. It is based on CORBA and uses IDL (interface definition language).

Any technical tester testing Windows software should know about COM/OLE/ActiveX.

Learning about COM

- "Bypassing the GUI," Brian Marick
  - Describes how to test using OLE/COM interface
  - http://www.testing.com/writings/bypassing-the-gui.pdf
- *Understanding ActiveX and OLE*, David Chappell
- *Mr. Bunny's Guide to ActiveX*, Carlton Egremont III

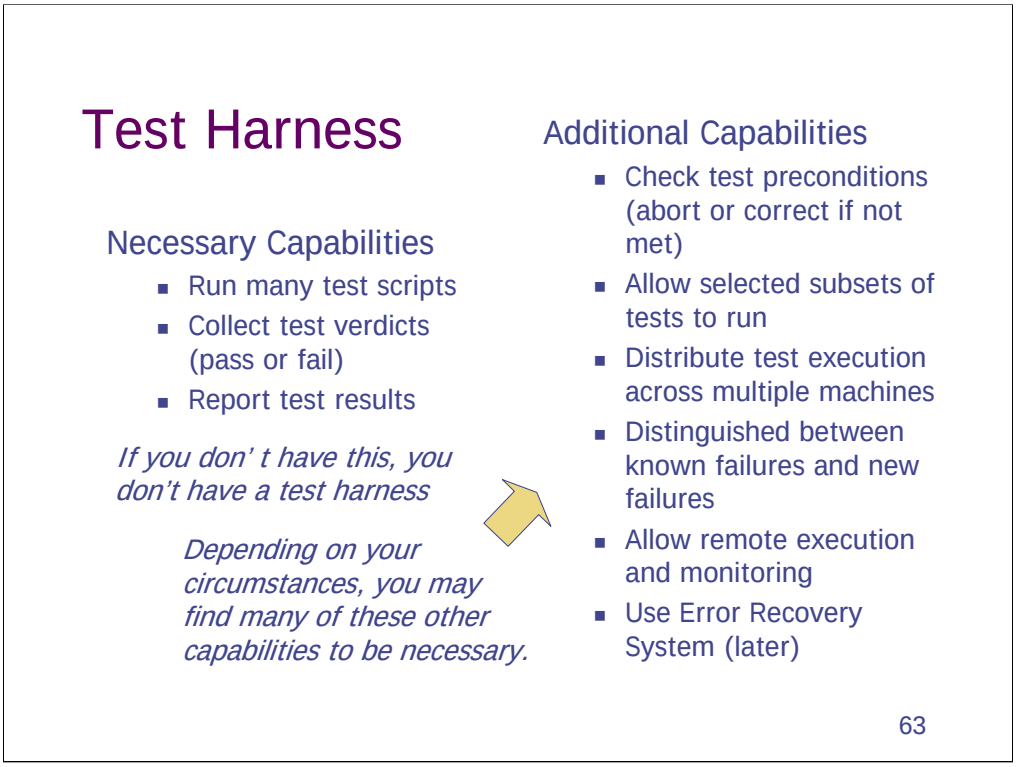
61

The Mr. Bunny book might not teach you much about ActiveX, but it's still pretty funny.

Test results can mean a lot of different things. It can mean whether the test passed or failed. It can mean the output that was generated when the test ran. It could also refer to log files or dumps.

The *test verdict* specifically refers to whether the test passed or failed.

Test harnesses will often handle both.



SEARCH is an anagram for the tasks of a test harness

Setup, Execution, Analysis (pass or fail?), Reporting, Cleanup, Help (documentation)

Setup Support

Tests typically have preconditions that must be met prior to being able to run the test. They may require that test records be pre-loaded or that a particular hardware configuration is present. Some test harnesses can check whether these preconditions have been met. If not they will abort the test and indicate that the test was blocked. Other test harnesses may actually be able to ensure that the precondition is met (say by loading the necessary data).

Marking Expected Failures

Some test harnesses allow tests to be marked as known failures. This means that there are known bugs that they encounter.

Some people think that it's nuts to include such tests in a test suite. If they fail and are expected to fail for long enough that it is worth the trouble to mark them so that they don't keep showing up in the failure reports, why not just remove them from the test suite? What good are they?

Types of Test Harnesses

- Different Test Harnesses have different ideas about what a test looks like.
- *Process-level harnesses* expect tests to be individual scripts. These are typically run in a separate process and can be in any scripting language.
- *Method-level harnesses* expect tests to be methods or classes in a specific language.
- Method-level harnesses can be adapted to work more like process level scripts.
  - You just write a method that loads a script.

64

Unit test harnesses are almost always method-level.

Process-Level Test Harnesses

- STAX
  - Uses STAF, thus multi-platform
  - Based on XML and Python
  - http://staf.sourceforge.net/getstax.php
- QM Test
  - Multi-platform, based on Python
  - http://www.codesourcery.com/qm/qmtest
- TET
  - Linux & Unix
  - http://tetworks.opengroup.org/Products/tet.htm
- Haste
  - Java-based
  - http://atomicobject.com/haste/

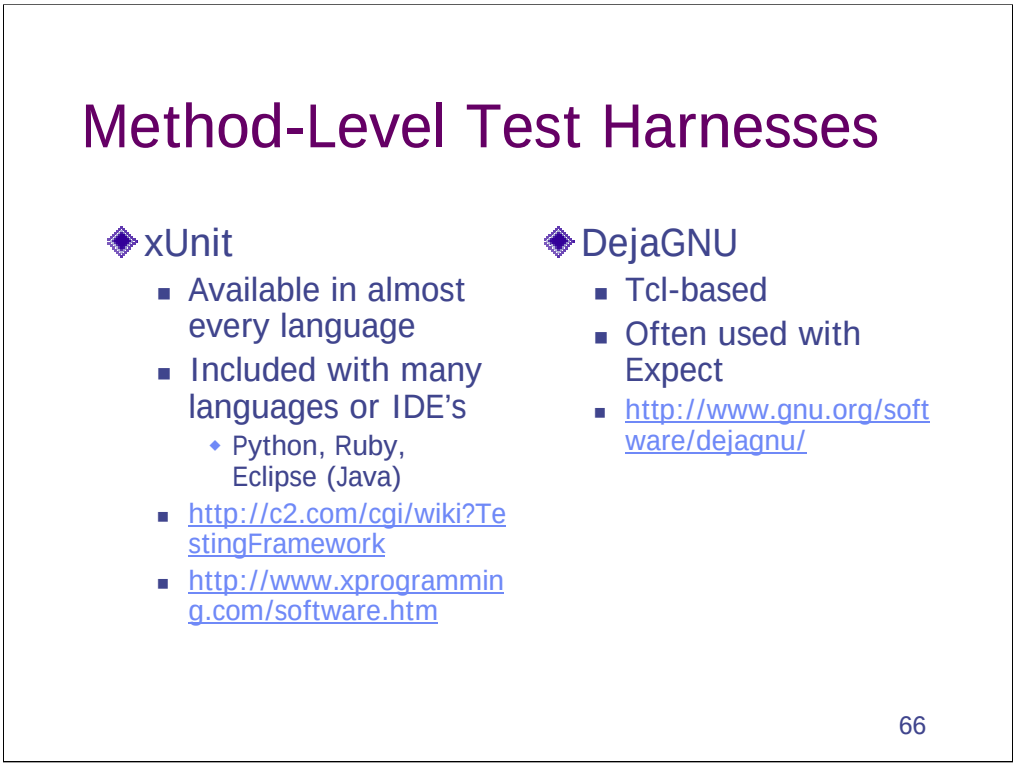
65

STAX is one of the richer services built on STAF. It is a test harness and test execution language with an XML-based grammar. The language has a number of unique logic primitives. For example, it has a parallel iterate command. This is like the for iterator found in many languages, except that each iteration is run concurrently in a separate thread. I'd never thought of making something like that a language primitive, although I'd recently added a similar feature to a python-based test suite. That took me three days to get all threading right. I could have probably done the same thing with STAX in a couple hours.

See STAF (in the slides ahead) for more info.

From the AWTA5 Report:

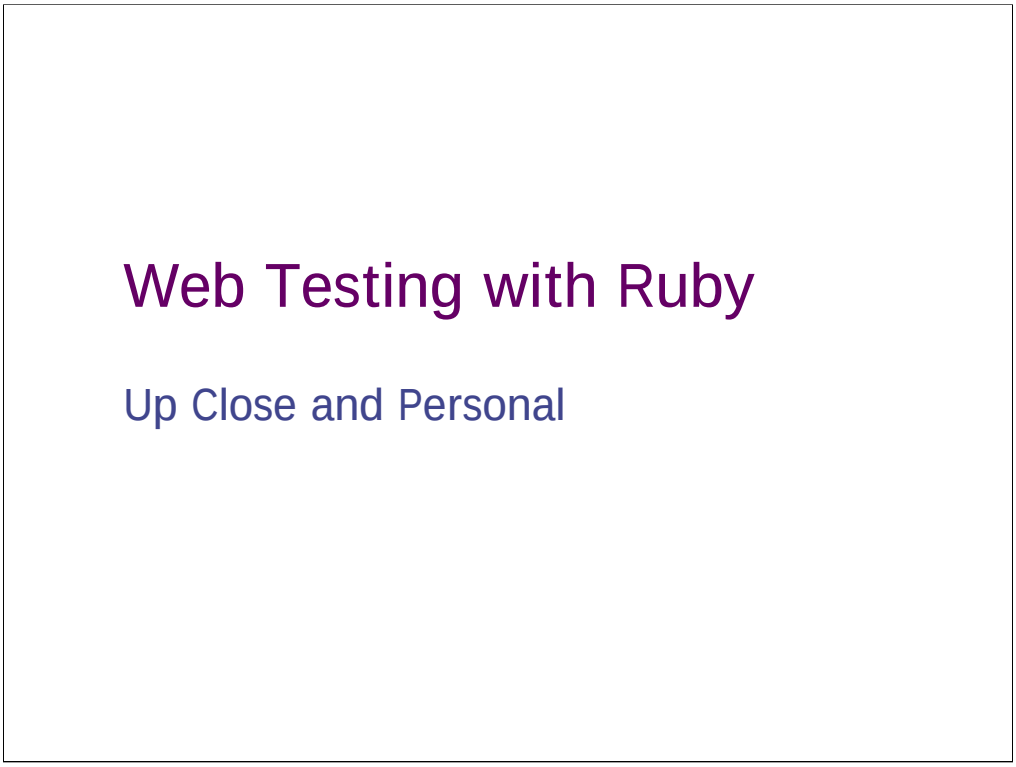
Carl Erickson presented Haste, a test harness for system tests in Java. With Haste, a new JVM is spawned for each test, preventing side effects from one test from affecting others. Of course, Haste itself runs in a JVM as well, so you get two JVMs. He demonstrated several tests, showing how they used other tools like Abbot and Marathon as the GUI drivers. He also showed how he uses "narc" classes. He's built a code-generator that creates special narc-versions of classes with public methods. This allows tests to call private methods and inspect private attributes without making changes to the base classes. He refers to this system of alternate testing classes a "narcitecture."



Some of the Interface Drivers already covered include a test harness, often JUnit.

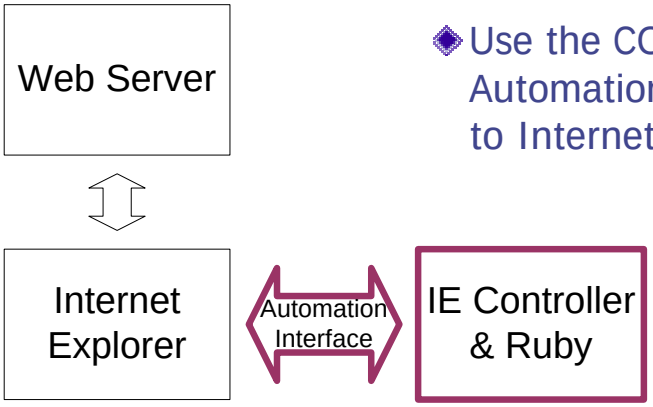
STAF (and STAX) are tools used widely at IBM that were made open source because of their use in IBM's Linux testing. STAF is a remote test agent. Many commercial testing tools include proprietary test agents that allow them to control tests on multiple machines. Now you can build the same capacity into your home-built automated systems. STAF has several services built in, including the ability to run executables, store local variables, and manage resource lists. It is designed as platform technology, allowing you to plug custom services into the framework. STAF runs on most operating systems: Windows, Unix, AS/400, MVS. The most obvious limitation is that it currently hasn't been ported to the MacOS and other BSD Unices, although Charles doesn't think that would be too hard. One of the great things about STAF is that the same services are available from a wide array of languages, including C/C++, Java, Perl, Tcl, Python and Rexx. There is even a command-line interface which looks for handy for analyzing test failures. STAF was the big hit of the AWTA5 workshop.

It's a bit of wonder that these tools ever made it public. This happened in the window of time between IBM's adoption of Linux and their acquisition of Rational. I expect that Rational is developing commercial testing tools that use STAF as their agent technology, just as they've already embraced Eclipse. STAF and STAX both come with extensive training materials.



Let’s look at an open source test tool, how it works, and how you could build something similar.

Browser Automation



- Use the COM Automation interface to Internet Explorer

Tests drive the browser

69

A snippet

```
require 'iec'
start_ie("http://localhost:8080")
get_forms[0].name = "bret"
submit_form(get_forms[0])
```

Time for a Demo

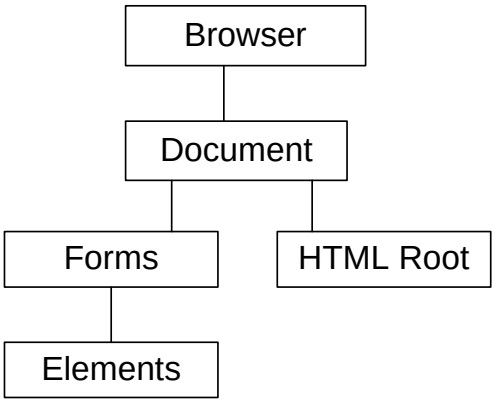
70

A longer example: start_stop.rb

DOM

- “Document Object Model”
- A non-proprietary standard for representing elements of a web page.
- Often used by client-side JavaScript.
- Supported by IE and Netscape and other browsers.
- IE provides access to the DOM via OLE.

DOM



72

IE Controller

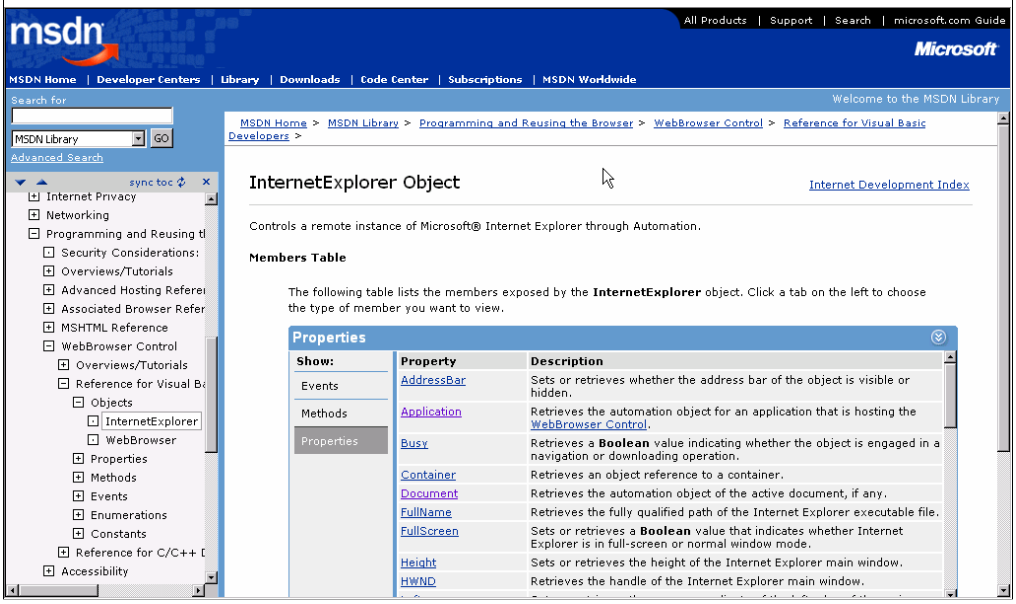
- Wiki
  - http://www.clabs.org/wtr/
- Mailing List
  - http://rubyforge.org/projects/wtr/
- Overview
  - http://www.rubygarden.org/ruby?IeController

Building Your Own

- Any decent language can call Internet Explorer's COM Automation interface.
- Many applications built with Microsoft technology have COM Automation interfaces.
- There are equivalent interface mechanisms for other technologies.

IE Automation Reference

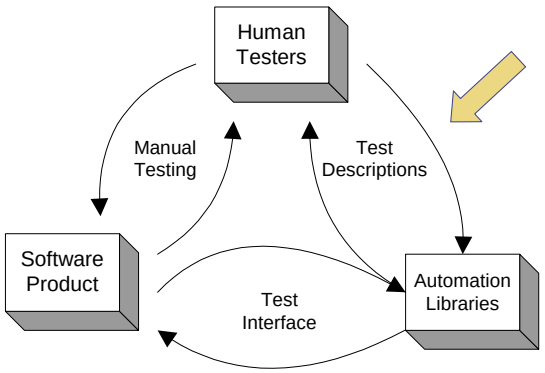
http://msdn.microsoft.com/workshop/browser/webbrowser/reference/objects/InternetExplorer.asp



Test Parsers & Generators

- Interpret and execute tests expressed in convenient form

Test Description Languages



Providing an effective means for describing tests which...

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

77

Test Parsers

- A test parser is a program that reads, interprets and executes a test written in a specific language or format.
- Data-driven testing is one example.
  - Tests are stored in a data format.
  - A program reads the data and executes the tests.
- Use test parsers when you don't want to write tests in the same language as your interface drivers.
- Parsers are available for common data formats for most programming languages.

78

You don't want to write a parser in a vendorscript.

Almost every general purpose programming language has libraries available for reading CSV (comma-separated values), TSV (tab-separated values), and DIF (data interchange format). There is even a library in Java for directly reading XLS (Excel) files.

Test Case	Window	Control	Method	Value
1000	MAINMENU	MAINMENU	SEL_MENU	Chart of Accounts
1000	MAINMENU	CHT_MENU	SEL_MENU	Enter Accounts
1000	CHT_ACCTS	ACCTNO	ENT_EDIT	100000
1000	CHT_ACCTS	ACCTNO	PRESSKEY	TAB
1000	CHT_ACCTS	ACCTDESC	ENT_EDIT	Current Assets
1000	CHT_ACCTS	ACCTDESC	PRESSKEY	TAB
		_ACCTS	STMTTYPE	PUSH_RB	ON
		_ACCTS	HEADER	CHECKBOX	ON
		_ACCTS	ACCTTYPE	LB_ITEM	Assets
		_ACCTS	OK	PUSH_PB	ON
		_ACCTS	MESSAGE	LOOKTEXT	Account Added

Name	Pass-word	Result
mfayad	xyz	login
dschmidt	123	expired
rjohnson	abc	reject

Go To	AddressBook			
New Address	Smith	John	1010 Main St	512-555-1212
Click On	Done			
Click On	AddressName	John Smith		
Verify Address	Smith	John	1010 Main St	512-555-1212
Change Address	SmithX	JohnX	1010 Main StX	512-555-xxxx
Click On	Done			
Click On	AddressName	John Smith		
Verify Address	SmithX	JohnX	1010 Main StX	512-555-xxxx

Here are some examples of different kinds of data-driven tests.

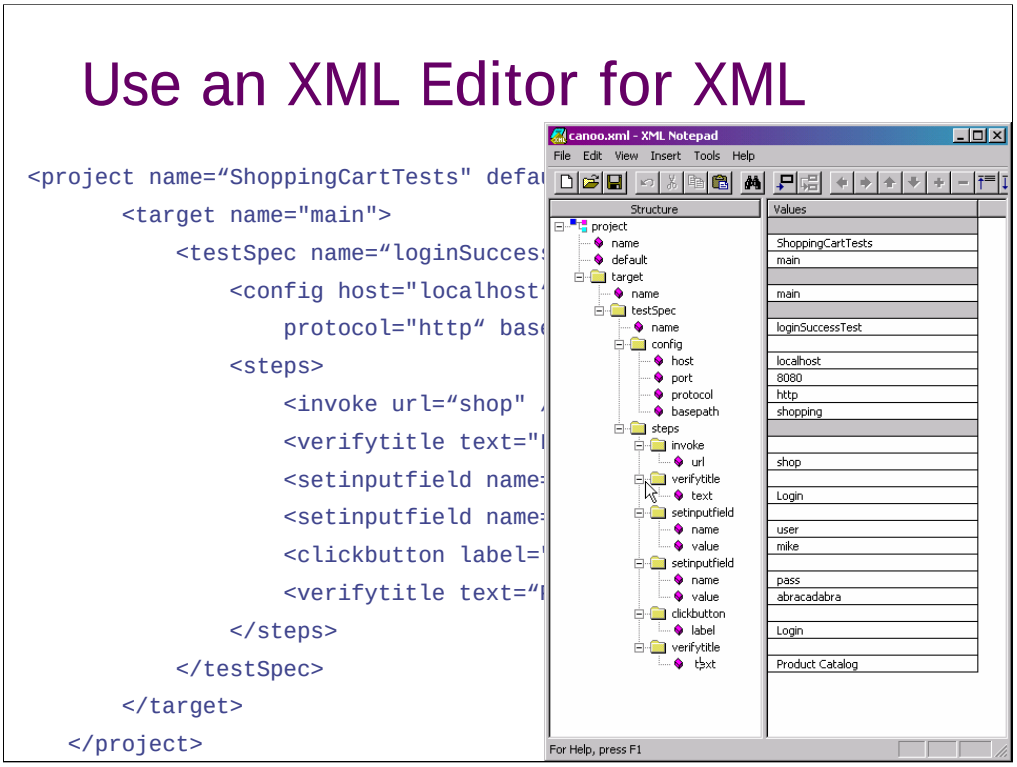
One is a simple table of data values.

One is made of a list of controls and the actions that should be taken with them.

The one at the bottom is an example of keywords or action-words. The first line indicates what the other data on the row is for.

Using XML

- Many of the tools listed with the interface drivers included XML parsers.
- XML parsers are available for all popular programming languages.
  - It's not hard to add support for an XML format yourself.
- Use an XML editor if you will be writing lots of XML.
- A DTD describes an XML format. An XML editor will use this to enforce the XML syntax.

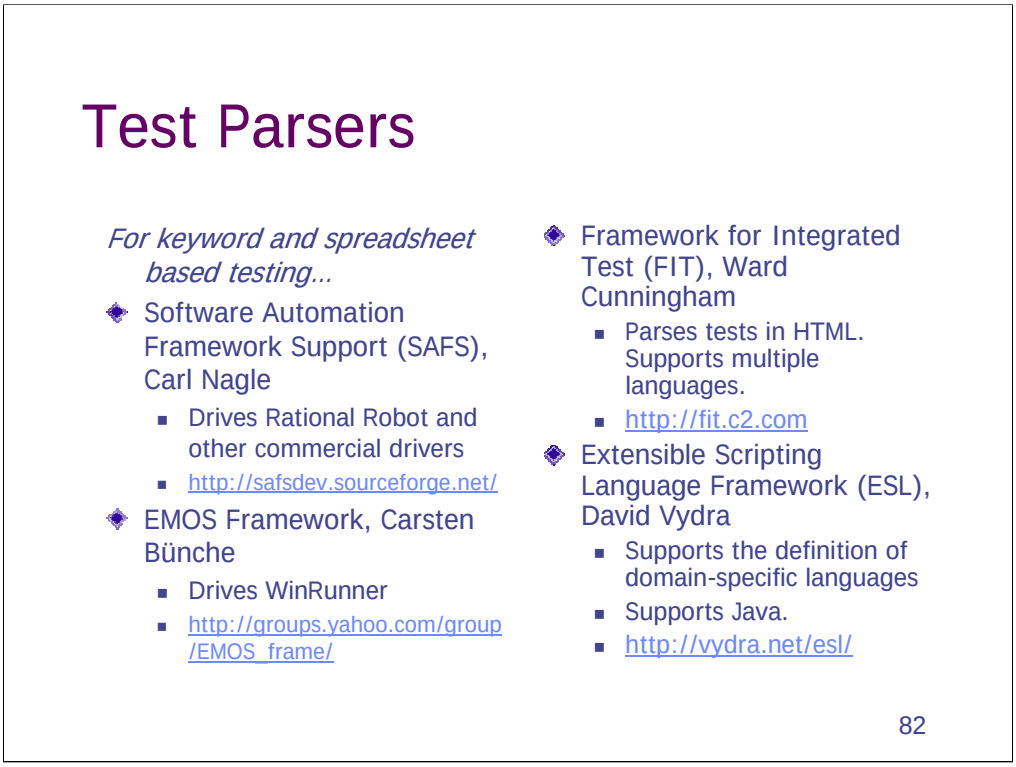


Here is some XML code (you saw this before: it's Canoo Web Test) overlaid by an editor editing the same code. Why looks easier to work with?

The editor shown here Microsoft XML Notepad

It's free (but not open-source)
http://www.snapfiles.com/get/xmlnotepad.html

If you are going to be using XML regularly, you'll want to get a better editor, which will probably cost you money. This one does not enforce the DTD (Document Type Definition). You will want one that does. Still it'll be cheaper and easier than building your own editor.



From the AWTA5 Report:

Carl Nagle, Bill Nasuti, Bob D'Antoni and Yuesong Wang presented SAFS, a data-driven engine. It parses tests written using keywords in a spreadsheet-based format and then executes them using a GUI test tool. It was initially developed for Rational Robot but has since been extended to also support Rational's RobotJ, which uses a completely different architecture (it is built using Java and Eclipse). In the process, they developed a tool-independent layer. As such, they believe it could be extended fairly easily to support other GUI test tools. The speakers represent three different companies (SAS Institute, Management Science Associates and Claritas). We got a good look at how and why test teams are creating new kinds of alliances in developing their automated suites. This was actually the first time that they had all meet face-to-face. The tool independence layer was implemented using STAF -- another tool presented at the workshop. Indeed some of us are wondering whether the test case parser and the tool independent layer could be logically separated, thereby giving tool independence to tests expressed in other languages or formats.

David Vydra presented ESL, Java-based parser technology for creating domain-specific languages. Keyword-driven tests are really just test expressed in a simple grammar. This grammar has one statement per line, with the first element representing a function and the remainder the arguments to the function. The more sophisticated keyword-driven frameworks (such as SAFS) also allow assignments of values to variables and additional loops and control logic. But if you are going to end up creating a grammar anyway, maybe you should consider using some of the standard tools and principles of computer language design. That's what David has done with ESL. With ESL, you can have a statement in a test that looks like "assert Customer with id = 100 has creditLimit <= 10,000.00". Is this an easier language for expressing tests? Does this language allow for the definition of more complex tests? I dunno, but now that we have an open-source implementation of this technology, i hope some comparisons will be made. Personally, I'm very interested to see what can be done with this.

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

83

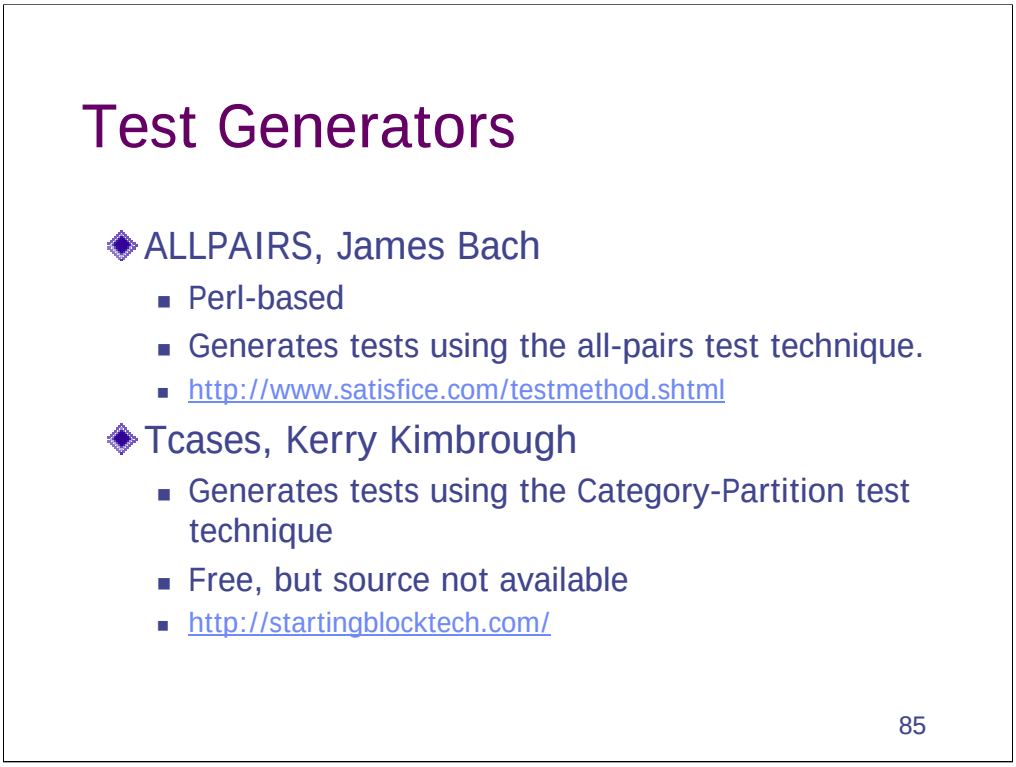
Test Description Languages

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



From the AWTA5 Report:

James Bach presented ALLPAIRS. Unlike the other tools, this is a test design tool. A classic testing problem is combinatorial explosion. There are so many different valid combinations of inputs to a system, that you just can't test all possibilities. ALLPAIRS uses an algorithm to create a subset of the combinations that meets a particular coverage requirement: that all pairs of combinations will be tested in one test case or another. This technique has been in the literature for some time, but it is tedious to do by hand. We actually included a description of the manual process in *Lessons Learned*, but the awkwardness of it lead James to develop this tool. This is the first easy-to-use open source implementation of the algorithm.

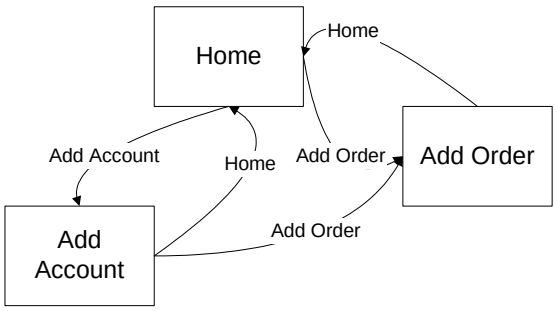
Jenny is another open-source tool that uses the same technique. It has a few more features than ALLPAIRS, but may be a bit harder to get started with. http://burtleburtle.net/bob/math/jenny.html

DGL is a data generation language, used to product test data according to a grammar.

http://cs.ecs.baylor.edu/~maurer/dgl.html

Model-Based Testing

- A state model of a web-based ordering system.



A small part of a model for an ordering system.

86

A state model consists of nodes and transitions.

It can be represented either as a graph, as here, or as a table.

Once you have a testing framework in a standard programming language, you can create models for generating tests that can be automatically executed.

Case Studies

Load Testing an Expense Report System

- Test Scenarios prototyped in Perl
  - Test actions only printed out
- Re-implemented in JavaScript for WebLoad
  - Programmer already knew JavaScript
- Data generated with Perl & loaded with Astra QuickTest
  - Company and Employee information had to be loaded in before the load test.

88

This is a real project that I worked on with James Bach. We used a mix of tools for this. The commercial tools were already at hand. We added the Perl code that filled in the gaps. At a certain point, we realized that we needed to preload the application with hundreds of company profiles and employee records. We used Perl to generate the data and then we loaded it in the database using QuickTest.

Today, I'd be more inclined to do everything in, say, Perl. Instead of reimplementing the test scenario logic in another language, we could have just extended the original prototype to actually run the tests. We could have used a Perl-based load generator, or used a protocol for the Perl code to communicate with a load generator.

One of the problems we had on the original project was that the JavaScript code had omitted some aspects of the Perl simulation. That problem would have been avoided if we weren't using so many languages.

Since we had Astra QuickTest, already, it was probably the smart thing to use to load the data. We just wrote a driver that read in the data (generated from the Perl scripts) and then entered it into the data entry screens. If we didn't have that, we might have used the wwwlib-perl library to directly submit the form data. Probably the best method would have been to have a stored procedure to directly load the data.

Testing Tape Backup Software

- Supported on many Unix flavors
- Had both command line and GUI interfaces
- Used Expect, Tcl and DejaGNU
- Used reservation system to obtain tape drives for testing

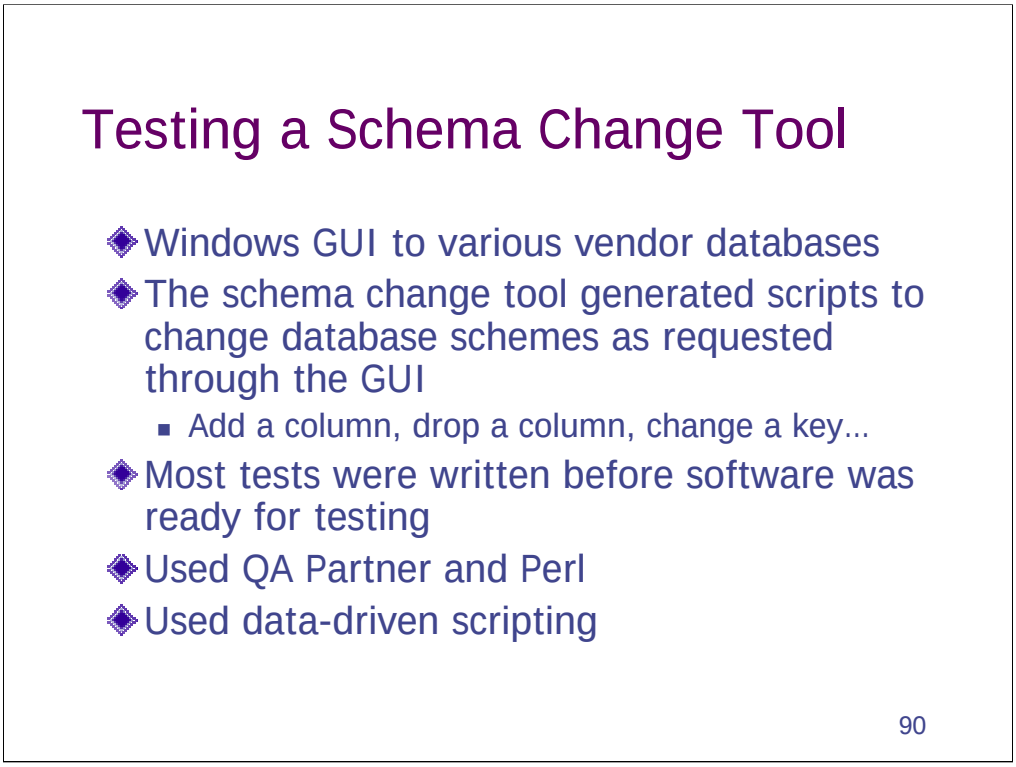
89

Expect worked very well for testing the command line interface.

We looked into automating the GUI interface. We had a copy of WinRunner. But we decided that it was better to test all the features from the CLI first and never actually got to WinRunner. Also, there were a lot of little bugs in the GUI that weren't going to get fixed, but which were going to complicate GUI automation.

We had to test the software using different kinds of tape drives. We didn't have that many. No dedicated test lab. So we had a homebrew database program to keep track of who was using which drives. The rule was that you couldn't even use the drive on your own system, unless you reserved it through this program. It had a command line interface. And you had to keep a writeable tape in it whenever you weren't using it.

If I was working on a similar system today, I'd write the reservation system using STAF.



Initially we designed a data-driven format to specify test cases. The QA Partner code read in these test cases and then executed them. This code got to be pretty hairy. One reason is that QA Partner was never really designed as a string-processing tool (unlike, say, Perl).

The data-driven format wasn't all that easy to understand. We found that the testers were first documenting their test cases in a more verbose format first that was easier for them to understand. We made a few changes to this verbose format to make it more standardized and then were able to write a parser in Perl that compiled this test documentation into the original data-driven format.

The biggest problem with this system was that it only generated the change scripts. Testers still had to manually verify that the change scripts were correct.

For more information, including samples of the different formats, see "Success with Test Automation," by Bret Pettichord, http://www.io.com/~wazmo/succpap.htm

Testing a Schema Change Tool: Test Driver Format

```
|A|dtbed101|E|TB||SA3|TB03||C|011|colname||NEW_CHAR_COL_LEN18
| |dtbed101|E|TB||SA3|TB03||C|011|datatype||CHAR(100)
| |dtbed101|E|TB||SA3|TB03||C|011|null||Y
| |dtbed101|E|TB||SA3|TB03||C|011|default||N
| |dtbed101|E|TB||SA2|TB03||C|011|colname||NEW_INTEGER_COL
| |dtbed101|E|TB||SA2|TB03||C|011|datatype||INTEGER
| |dtbed101|E|TB||SA2|TB03||C|011|null||Y
| |dtbed101|E|TB||SA2|TB03||C|011|default||N
| |dtbed101|E|TB||SA3|TB02||C|035|colname||NEW_LOB_COL_AT_END
| |dtbed101|E|TB||SA3|TB02||C|035|datatype||CLOB(5K)
| |dtbed101|E|TB||SA3|TB02||C|035|null||Y
| |dtbed101|E|TB||SA3|TB02||C|035|default||N
| |dtbed101|E|TB||SA3|TB02||C|035|logged||Y
```

Testing a Schema Change Tool: Documentation Format

```
TEST CASE ID: dtbed101
EDIT TABLE: SA3.TB03

ADD COLUMN(S)
Position  NAME                TYPE       NULLS     DEFAULT   FOR BIT DATA
          LOGGED    COMPACT
11        NEW_CHAR_COL_LEN18  CHAR(100) Y         N
Note: Column name is of maximum length and is of type char.
--------------------------------------------------------------------------------
EDIT TABLE: SA2.TB03

ADD COLUMN(S)
Position  NAME                TYPE       NULLS     DEFAULT   FOR BIT DATA
          LOGGED    COMPACT
11        NEW_INTEGER_COL     INTEGER    Y         N
Note: Column is of type integer.
--------------------------------------------------------------------------------
EDIT TABLE: SA3.TB02

ADD COLUMN(S)
Position  NAME                TYPE       NULLS     DEFAULT   FOR BIT DATA
          LOGGED    COMPACT
35        NEW_LOB_COL_AT_END  CLOB(5K)  Y         N         -         Y
          -
Note: Column is of type clob.  Logged is the default.
```

92

Testing Data Movement Monitoring

- Data moved reliably & securely between all kinds of systems
- Had to test monitoring tool
- Monitor was in Java on Windows
- Used Python to generate data movement jobs
- Used WinRunner to test monitor
- Also used Python for load testing the servers

93

The company developed a line of products to support data movement on a variety of platforms, including Windows, Unix and mainframe. Think of FTP on steroids. One of the products was a monitor that showed you what was happening with the others.

The monitor had been unit tested using mock servers. But system testing had to be done with the real data movement servers. Although WinRunner could test the monitor directly, It couldn't generate the loads. The servers used a proprietary protocol. The company had a Java library for this protocol. We used Python and this library to write tests that would create the required data movement jobs on the different servers.

We extended Python's xUnit to create a test harness that would load the test scripts. That way testers could also run their scripts outside their harness. This made them easier to create and debug.

Using Python's thread support, we were able to extend the test harness to submit multiple jobs at the same time. This quickly gave us very effective load tests.

Your Homebrew Project

A Language-Based Approach

- Select a general purpose programming language for your test libraries.
- Maximize reuse.
- Use the programming skills your team already has.
- Choose tools that will work with your language choices.

95

Be the Master of Your Tools

- Your tools don't know how to test. You do.
- Use tools that can be combined with other tools.
  - Commercial tools often resist this.
- Avoid depending on node-bound tools.
  - Everybody wants to be able to use your tests. That's why you wrote them.
- You need to be able to debug your tools when they stop working.
- But if you have commercial tools, use them when appropriate.

96

Understand the Motivations of Test Tool Vendors

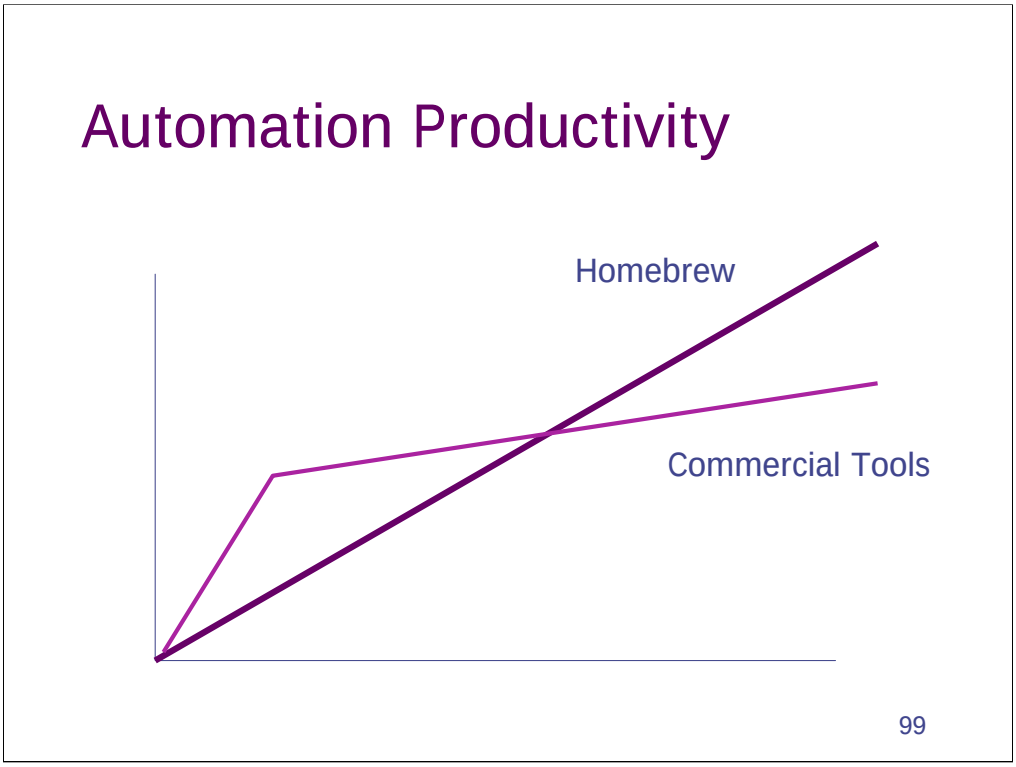
- Want you to think they are experts in testing.
  - Guess what? They're not.
- Encourage belief in tool magic.
- Achieve lock-in when you your test library is in their language.
- Focus on improvements that people will see during the eval period.
  - Enhancements for power users go on the back burner.

97

Test tool vendors regularly bill themselves as testing experts. I've worked closely with a lot of vendors. In general, I have been surprised to find that they don't have any more expertise or insight into effective software testing than is found in other software companies. People who are presuming that they have knowledge and expertise to share are misguided. (And I'm not even talking about the lies their salesmen will tell to sell product.)

Commercial Tools	Homebrew
Vendorscripts discourage developer involvement	Rich languages encourage developer involvement
You must beg vendors to fix bugs, support new platforms	You can fix bugs and add support yourself when necessary
Compatibility interfaces are secret	Compatibility interfaces are open and modifiable

98



Homebrew solutions often take more time to get ramped up. But one they are up and running, people see continued increases in productivity due to richness of the languages and the large number of libraries.

Commercial tools are optimized for out-of-box experience. People can get get productive with them very quickly. But with time the limitations of the tool limit what can be done.

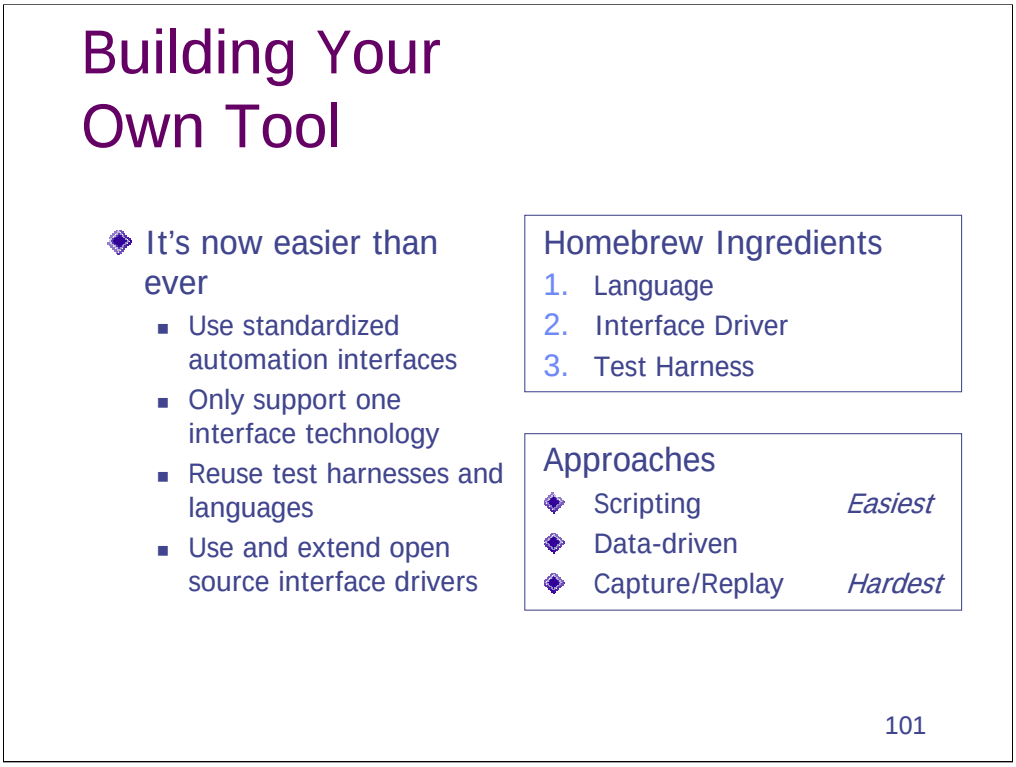
Building Your Own Testing Framework

1. Choose a language that is the best fit for your staff.
2. Find or create drivers for the interfaces your product has.
3. Work with developers on testability and alternate testable interfaces.

It's now easier than ever

- Use standardized interfaces
- You only need to support the interfaces you are using
- Use open-source test harnesses and languages

100

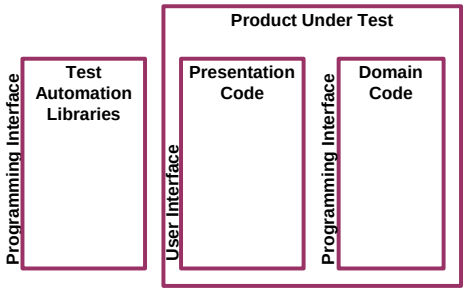


This is now a major part of my consulting business.

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

102

Excel: Product accuracy was a major requirement and depended on ability to execute and re-execute tests. Therefore the product was designed with a test interface that closely mapped the user interface. This interface was latter exposed to users and is presently accessible through VBA.

Xconq: Public domain game system has multiple user interfaces (X, curses, mac, tcl/tk) as well as a command line interface specifically for testing. http://dev.scriptics.com/community/features/Xconq.html

InstallShield: Response files can be used for a silent install. This is an excellent testability interface available to any product that uses install shield.

Autocad & Interleaf: Both products can be extended using Lisp. In both cases this was used to help automate tests.

Tivoli: Products have command line interfaces as well as GUI interfaces. The command line interfaces were of great advantage for automation.

Client/server interfaces: You can test a web server directly via http rather than using (and automating) a browser. Another option.

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!

103

More Open-Source Test 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

104