Getting Started With Automated Testing

Pacific Northwest Software Quality Conference
October 11-13, 2004
www.PNSQC.org

Michael Kelly
www.MichaelDKelly.com

Introduction

Focus will be on developing a strategy for automation and putting together an automation team.

Depending on your context, some of the may be more valid or necessary than others.

There is no particular order in which the steps need to be executed.

The steps discussed apply primarily to teams of three or more people.

Not geared to a specific test tool or tool vendor.

Introduction

Automated testing is the use of tools to assist with a testing effort.

The following is a small sampling of different types of automation available:

- Load and performance testing
- Installation and configuration testing
- Testing for race conditions
- Endurance testing
- Helping the development effort with smoke tests and unit tests
- Analyzing code coverage and runtime analysis
- Automation of test input generation
- Checking for coding standards and compliance
- Regression testing
- many more…

Automated testing is a subset of the overall testing profession.

Introduction

Main focus on scripted automated testing

- Performance testing
- Regression testing
- Unit testing
- Other type of scripted testing

There is an overlap of the skill set for those who make good testers and those who make good automated testers.

Automated tester should be a tester first. They should be skilled in:

- black box testing techniques
- white box testing techniques
- performance testing
- exploratory testing
- and any of the hundreds of other types of testing

1. Set Goals for Automation

One of the problems most teams encounter is that they only focus on a few of the more popular types of automated testing

Setting goals for automation can affect what you automate.

- Do you want to find bugs?
- Do you want to establish traceability for some sort of compliance?
- Do you want to support the development team?
- Do you want to ensure that there are no changes in the software?

1. Set Goals for Automation

Your software development lifecycle and the skills of your testing staff will also affect what you choose to automate:

- Are you developing in an XP lifecycle with programmers also testing? This would probably lead to more automation.
- If all of your testers are junior, you probably will use very little automated testing or automated tests that are shallow.

If your goal is to support your developers, select tools and training to support that goal:

- You will need testers who know how to program and who can communicate with developers in their language.
- They will need to look at source code, configure environments, and will probably spend a good deal of time working side-by-side with developers debugging and troubleshooting.

In contrast, if your main goal is to support refactoring and change:

- You will have a team with experience developing regression scripts and need to have team members with both strong business knowledge and strong testing skills.
- They will need to know how to create scripts that look for changes and will need to know the business in order create scripts that look for underlying changes that may be taking place.

1. Set Goals for Automation

A common mistake when automating for the first time is biting off more than you can chew:

- This typically leads to missing deadlines
- This can force your team to work unrealistic hours
- This can demotivate your testing team
- This can make testing look bad in the eyes of the rest of the development team

Use your goals to set the strategy and scope for your automation.

When deciding what to automate for your first time, start with small milestones.

- Start with testing simple functionality.
- Start with just one virtual user and set your goal at a low number (no more than twenty).
- You should be able to use record-and-playback features to perform basic testing or you can do some basic scripting.

Once you have determined your goals, you will need to start thinking about what kinds of skills you will need to have on your team.

2. Have at Least One Programmer on Your Team

I recently saw a company do a comprehensive analysis of three market-leading enterprise level test tools:

- They gave little to no thought about:
  - who would be using the tool
  - their technical experience
  - where they could get training
  - and how much it would cost
  - if they had past experience with one of the other tools
  - etc…

A tool is and always will be, only a tool.

2. Have at Least One Programmer on Your Team

The most important thing you can do is select the right team.

To ensure efficient well-planned automated testing, you will need to have at least one experienced programmer in your testing-automation group.

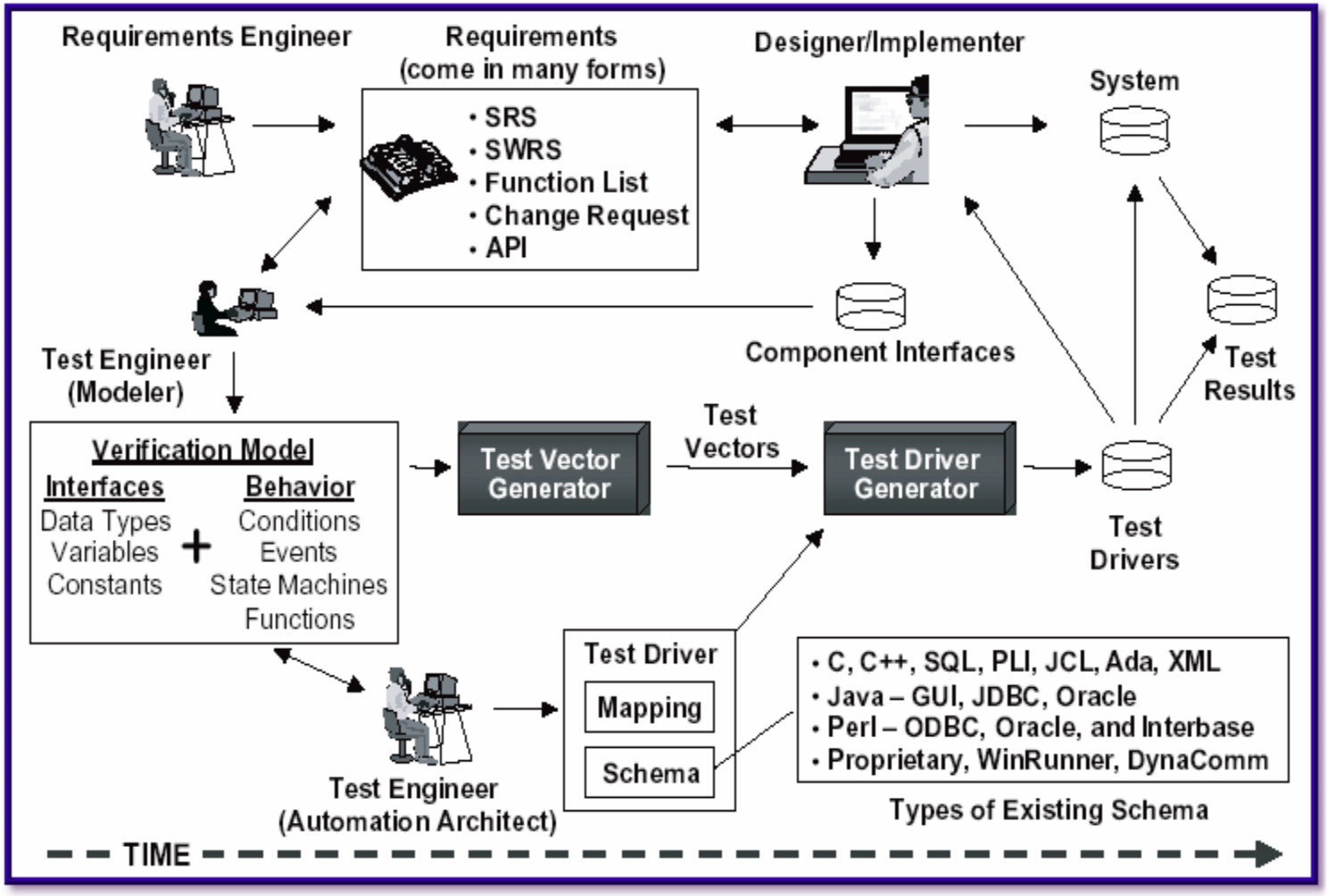
Record-and-playback features only offer quick solutions for the most common tasks and controls. For advanced automation of any kind or for any custom controls, you'll need to be able to write your own *maintainable* code.

Beware you don't employ only programmers. All team members should be testers first. Skilled in the mental arts of testing as well as armed with the ability to code and design test systems.

2. Have at Least One Programmer on Your Team

Experienced Programmer:

- Automated testing *is* code development.
- You will want to build modularity into your scripts.
- You will want to create some custom functions, extend the test tools, and potentially read test data from databases or data files.
- Depending on the tool and language, encapsulation, object orientation, or some other method may be used to make the code more maintainable or easier to use.
- If you are looking at providing code analysis services (performance profiling, code coverage, runtime analysis), you will need someone who knows how to read the code-under-test and who can ask the development team the right questions.



Graphic taken from "Interface-Driven, Model-Based Test Automation"
by Dr. Mark R. Blackburn, Robert D. Busser, Aaron M. Nauman

2. Have at Least One Programmer on Your Team

Record and Playback:

- Even if the scripting environment you are using is not Java or C++ (which it may very well be), you're still building systems of scripts, data files, and libraries.
- That means employing programmers, not manual testers who learn to code as they go.

```
import resources.CalculatorExample_AddHelper;

/**
 * @author Michael Kelly
 */
public class CalculatorExample_Add extends CalculatorExample_AddHelper
{
    /**
     * Script Name   : <b>CalculatorExample_Add</b>
     * Generated     : <b>Oct 5, 2004 8:18:49 PM</b>
     * Original Host : WinNT Version 5.1  Build 2600 (S)
     *
     * @since  2004/10/05
     * @author Michael Kelly
     */
    public void testMain(Object[] args)
    {
        startApp("calc");

        //Add 5 + 5 and verify the result is 10
        _5button().click(atPoint(27,19));
        _button().click(atPoint(20,13));
        _5button().click(atPoint(9,18));
        _button2().click(atPoint(19,12));
        _10_selectedTextVP().performTest(2.0, 20.0);
        calculatorwindow(ANY,MAY_EXIT).click(CLOSE_BUTTON);
    }
}
```

3. Get Familiar with the Tools

The scope of your automation effort will depend heavily on the tools you use and have access to.

Most automated testing tools are designed for unit testing, some sort of regression testing, or performance testing.

There are now many good open source tools now available.

Regardless of what tool(s) you select, you will need to be sure that you spend time learning how to use them properly.

- Go through the tutorials that are provided
- Read through whatever documentation is available
- Attend a training class
- Hire a training consultant to come in and spend some time with you

3. Get Familiar with the Tools

- Disk imaging tools
- File scanners
- Macro tools
- Memory monitors
- Environmental debuggers
- Requirements verifiers
- Test procedure generators
- Syntax checkers/debuggers
- Runtime error catchers
- Source code testing tools
- Environment testing tools
- Static and dynamic analyzers
- Unit test tools
- Code coverage tools
- Test data generators
- File comparison utilities
- Simulation tools
- Load/Performance testing tools
- Network testing tools
- Test management tools
- GUI testing tools
- Any scripting language (Ruby, Pearl, VB, etc…)

3. Get Familiar with the Tools

Open Testware Reviews:
http://tejasconsulting.com/open-testware/

- A source of information about freely available test tools.

Testing Tool Information:
http://www.grove.co.uk/Tool_Information/Choosing_Tools.html

- Short Sharp Advice about how to choose a testing tool.

Test Tool Evaluation Center:
http://www.testtoolevaluation.com/index.asp

- An online resource with a lot of info and wizards for tool comparisons.

4. Set Some Standards

This is just as important in testing as it is in conventional software development.

Your test system will develop more rapidly and will be easier to maintain if you establish and enforce standards.

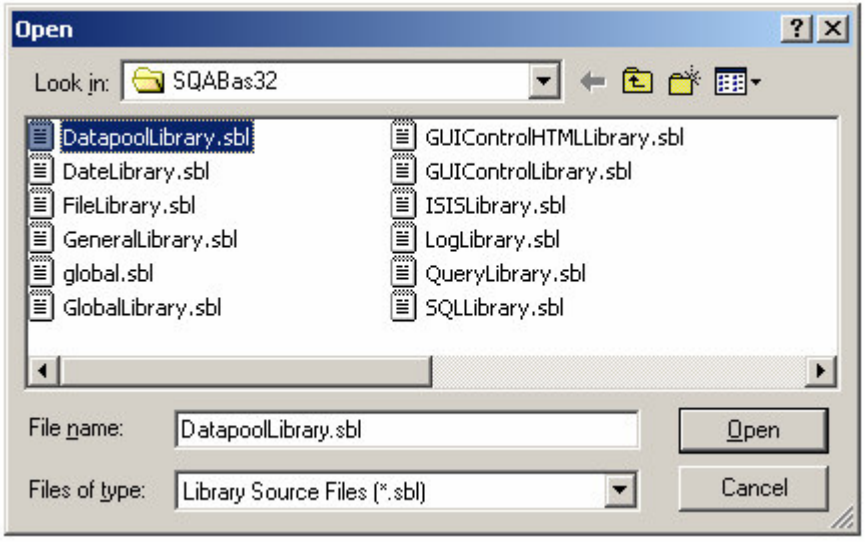
Having these standards documented will also allow people new to the project team to come up to speed faster.

4. Set Some Standards

Naming standards

- Scripts
- Test logs
- Directory structures
- Data structures
- Verification points

Name	Script Type	Description
CLAIMS-1101-Admin-Preferences	GUI	Preferences in user admini...
CLAIMS-1102-Admin-Settings	GUI	Settings in user administration
CLAIMS-1103-Admin-Alerts	GUI	Alerts in user administration
CLAIMS-1201-Reports-Month End	GUI	Month end report
CLAIMS-1202-Reports-Annual	GUI	Annual report
CLAIMS-1203-Reports-Customer	GUI	Customer report
CLAIMS-1301-Scenario-Customer 1	GUI	Typical data for customer 1
CLAIMS-1302-Scenario-Customer 2	GUI	Typical data for customer 2
CLAIMS-1303-Scenario-Customer 3	GUI	Typical data for customer 3
CLAIMS-1304-Scenario-Customer 4	GUI	Typical data for customer 4
QUOTE-1101-Rating-Scenario1	GUI	Data for scenario 1
QUOTE-1102-Rating-Scenario2	GUI	Data for scenario 2
QUOTE-1103-Rating-Scenario3	GUI	Data for scenario 3
QUOTE-1104-Rating-Scenario4	GUI	Data for scenario 4
QUOTE-1201-Policy-Automotive	GUI	Tests for automotive lines ...
QUOTE-1202-Policy-Home	GUI	Tests for home lines of bu...
QUOTE-1203-Policy-Watercraft	GUI	Tests for watercraft lines ...
QUOTE-1204-Policy-Renters	GUI	Tests for renters insuranc...
QUOTE-1205-Policy-Life	GUI	Tests for life insurance line...
QUOTE-2101-Reports-Billing	GUI	Billing reports
QUOTE-2102-Reports-Administrative	GUI	Administrative reports
QUOTE-2103-Reports-Month End	GUI	Month End reports
QUOTE-2104-Reports-Billing Detail	GUI	Billing Detail reports



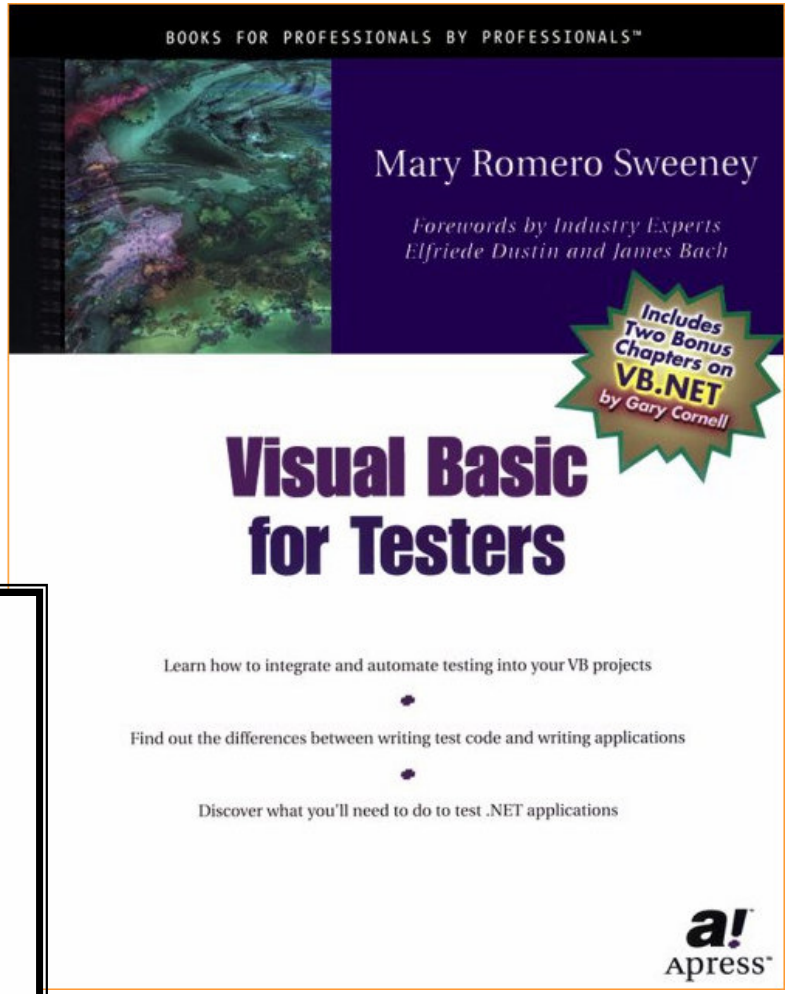
On two of my last three projects:

- over 1,000 scripts in each project
- over 5,000 data structures for each project

4. Set Some Standards

Coding standards:

- Steal the standards used by the development staff
- Go online and find some
- Customize them to fit your needs and the needs of your team



The GNU coding standards, last updated October 1, 2004.

- Preface: About the GNU Coding Standards
- Legal Issues: Keeping Free Software Free
- Design Advice: General Program Design
- Program Behavior: Program Behavior for All Programs
- Writing C: Making The Best Use of C
- Documentation: Documenting Programs
- Managing Releases: The Release Process
- References: References to Non-Free Software or Documentation
- Copying This Manual: How to Make Copies of This Manual
- Index

http://www.gnu.org/prep/standards/standards.html#Preface

4. Set Some Standards

Environmental standards

- Ensure that the computers you use all have the same operating system, RAM, hard drive space, and installed software configurations.
- The only differences should be the specific differences you are looking for in your configuration testing.

Procedures for error and defect tracking

- Describe how to log errors in test scripts
- Describe how to submit defects via your defect-tracking tool
- Describes how to code workarounds into scripts, and remove them after a bug is resolved

5. Establish Some Baselines

Figure out what the bare minimum tests are for that type of testing to be successful and implement them in very small, very simple chunks.

You will use these baselines as the foundation for what you will be implementing going forward.

Have some sort of archiving or source control and can always come back to these simple tests if you need to.

Once you have established your baselines you will be better prepared to start looking at more advanced methods of scripting (or data gathering, or scenarios, etc…).

5. Establish Some Baselines

In past projects I have established the following baselines:

- For **regression testing**: a baseline set of scripts that test basic functionality in the system using the most commonly used paths through the application.
- For **performance testing**: a baseline set of scripts that target each standalone service for the application that run successfully with one virtual user.
- For **code coverage analysis**: try to develop a series of tests that will allow you to get some level of coverage in each major module of your application.
- For **source code checking** tools: try to get a small subset of the application code to pass the verifications and modify/refine the rules as needed.
- For **establishing traceability**: take a small series of test cases and, using your test management tool, establish traceability from the manual/automated scripts, to the test cases, to the requirements. Generate a simple report that shows this traceability in a usable format.
- For **test data generation**: generate a subset of the overall data you will need for testing and verify the data's correctness and randomness.

6. Look for Ways to Modularize Your Scripts

Look through the code in the scripts for repetitive calls or other common actions. The goal is to make maintenance is as easy as possible.

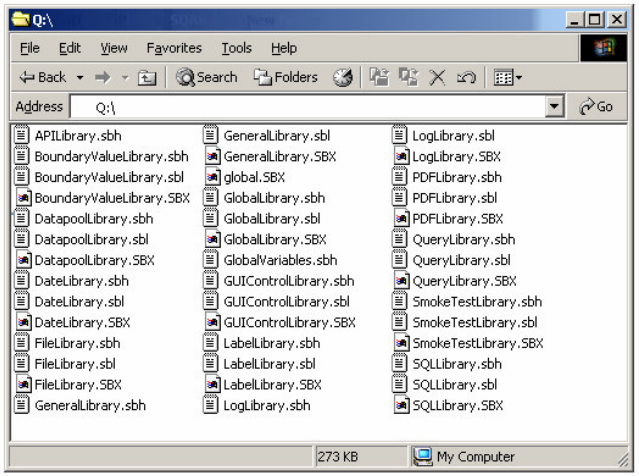
For ideas on how to modularize your scripts:

- Look at user communities for the tools you use
- Read literature on the topic from any of the major testing websites
- Talk with your development team
- Hire in a consultant to train your staff on what to look for.

6. Look for Ways to Modularize Your Scripts

Just like any piece of software, a script *will* cost you more to maintain than it will to create.

More than likely, you'll carry these over to following projects, and they'll evolve and change as you do.



I use the same code for:

- SQL
- File access
- Java and HTML controls
- Tool specific workarounds
- Windows and Internet Explorer APIs

If you make is generic enough, you can even make it a stand alone automation tool:

- Screenshot viewer
- Log file parser
- Data generation tool

Opensourcetesting.org

Open source tools for software testing professionals

7. Use Data Structures & Data-Drive Techniques

Effective and cost-efficient automated testing is data-driven.

- Offers the greatest flexibility when it comes to developing workarounds for bugs
- Decreases time spent maintaining code
- Allows for the fastest development of large sets of test cases

Most likely, your tool will have some method for implementing this. If not you can use:

- Spreadsheets (any csv file)
- Databases (any ODBC source)
- DAO/ADO objects
- Web services

7. Use Data Structures & Data-Drive Techniques

Data selection

You will need to select the data that either drives the navigation of your application, represents the data that gets entered into your application, or both.

- **Based on risk:** The number one motivator for test data selection should be risk.
- **Based on requirements:** Select data that will allow you explicitly test a requirement or a set of requirements (or an operating environment).
- **Based on availability:** The idea here is that if the data is easily *accessible*, it is *usable*, and it is *meaningful*, then you might want to include this data in your testing. Look for sources where you could potentially save some time and money by getting extra mileage out of work you have already done or data that is readily available.
- **Based on production data:** Production data can be one of the richest resources for scenarios for automated testing, both because the data is representative of real scenarios the application will face, as well as it will most likely provide a high number of different scenarios.

  NOTE: There are potentially some legal issues surrounding the use of production data (HIPPA, Sarbanes-Oxley, Gramm-Leach-Bliley Act, etc…), especially if you outsource some of your testing.

- **Random Data Generation:** Many tools include test data generators. Random data generators can be especially helpful in generation large sets of data.

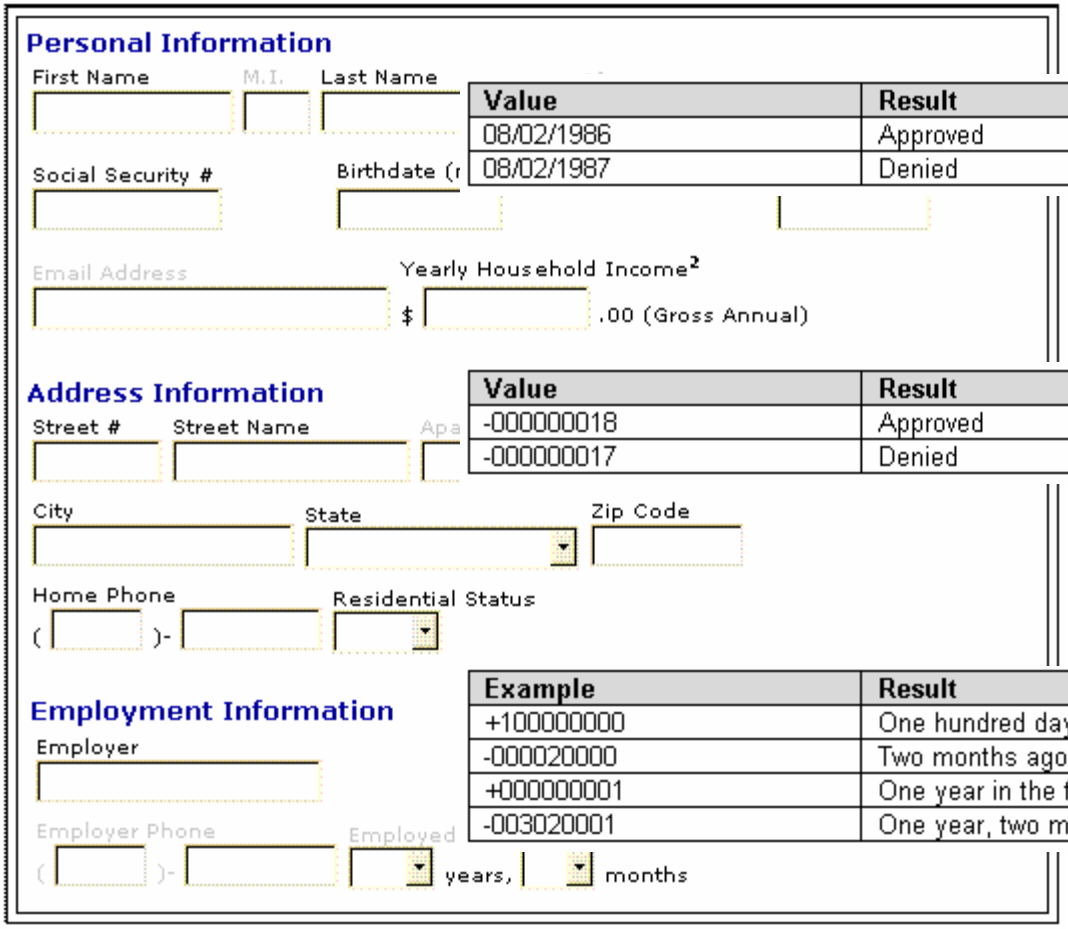
7. Use Data Structures & Data-Drive Techniques

Design for repeatability

With traditional automation (GUI-based automated regression testing), return on investment is often realized after the automated test has been executed several times.

- **Database re-shoot:** You need the ability to rebuild the baseline data
- **Stand-alone data:** You don't want data that relies on previous data
- **Unique data:** You will want to be sure that one set of test data is distinguishable from another.
- **Reverse delta processing:** Reverse delta processing is a method of implementing dates within your test data so that they never age. Take for example the following screen shot of an application used to processes credit card applications.
- **Default vs. test data:** By setting up a default dataset you can save space and time on the creation of test data as well as simplify the test dataset making it easier to read and problem shoot.

Reverse Delta

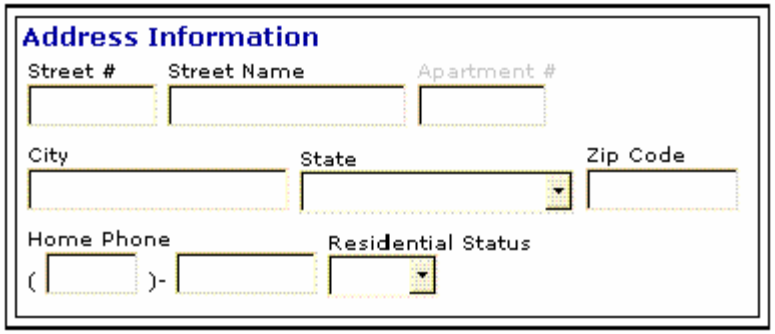


Value	Result	Comment
08/02/1986	Approved	18 years old
08/02/1987	Denied	17 years old

Value	Result	Comment
-000000018	Approved	18 years old
-000000017	Denied	17 years old

Example	Result
+100000000	One hundred days in the future.
-000020000	Two months ago.
+000000001	One year in the future.
-003020001	One year, two months, and three days ago.

Default vs Test Data



Default Data:

Street #	Street Name	Apart #	City	State	Zip Code	Phone	Status
1234	Washington Blvd		Penn Yan	NY	14527	555-345-4776	Owner

Test Data:

Street #	Street Name	Apart #	City	State	Zip Code	Phone	Status
			Berne	IN	46711		
			Big Lake	TX	76932		
			Yakima	WA	98901		
			Wellston	OH	45692		

8. Document Everything (…that it makes sense to document…)

Find out what minimum documentation is required for your context.

After you have met the minimum requirements:

- Document why you designed things the way you did.
- Document what each module does, and what each function in it does.

This documentation is often most useful as:

- Training material
- Future technical reference (much like an API)
- Helps you keep track of lessons learned

Sometimes documentation is the only thing that can save project scripts that no one has worked on in a while.

Review

1. Set goals for automation.
2. Have at least one experienced programmer in your testing-automation group.
3. Get familiar with your tools.
4. Develop standards for your team.
5. Establish some baselines.
6. Modularize and build reusability and maintainability into your scripts.
7. Use data-driven testing techniques whenever possible.
8. Document what makes sense to document.

Review

- Focus on the clarity and simplicity of your goals
- Focus on your understanding of the tools
- Focus on the makeup of your team
  - It has been my experience that the makeup of the team is the most influential factor in an automation efforts success.
  - Test automation requires a balance between programming knowledge, testing skill, and business knowledge.
  - Incorporating all of those assets into your team will be the most important step of all.
  - Tools will do nothing if people can't use them or don't know how to test to begin with.

Thank You

Questions?