

[Presentation Notes](#)
[Paper](#)
[Bio](#)
[Return to Main Menu](#)

P R E S E N T A T I O N

T2

Thursday, October 29, 1998
10:30AM

THE DIFFERENCES BETWEEN TEST AUTOMATION SUCCESS AND FAILURE

Elisabeth Hendrickson
Quality Tree Consulting

International Conference On
Software Testing, Analysis & Review
October 26-30, 1998 • San Diego, CA

The Difference Between Test Automation Success and Failure

a tale of two automation projects

Elisabeth Hendrickson
Quality Tree Software, Inc.
esh@qualitytree.com

What is Failure?

- Wasted Time
- Wasted Money
- Inaccurate Results
- Demoralized Team
- Overall Reduced Productivity
- Lost Opportunity

What is Success?

- Overall Cost Savings
- Improved Testing
- Shortened Software Development Cycle
- Reliable Results
- Process in Place for Future Success

Two Projects: a Failure and a Success

Failed Project

- 1993 - 1994
- Dedicated team of 4 people including 3 automators & a lead (me).

Successful Project

- 1996 - present
- Dedicated team of 3 people. The lead (me) did as much programming as anyone else.

Both Projects

Automating a complex client/server system with an open architecture (worked with Sybase, Oracle, as well as a variety of network operating systems).

Project-Level Differences

- Experience level of leadership
- Goals of the projects
- Communication
- Readiness to automate
- Role of the automated testing team

Characteristics of Leadership

Failed Project

- Executives expected immediate payback.
- QA Manager had unrealistic expectations.
- Automation lead (me) inexperienced in leadership and automation.

Successful Project

- Different executives were open to having their expectations reset.
- Different QA manager with more automation experience.
- Automation lead (me) got a clue.

Goals of the Projects

Failed Project

- Stated goal: “Automate Everything”
- Unstated goal: “Reduce number of testers needed.”
- Goals not measurable.

Successful Project

- Stated goal: “Save manual testers time and improve testing coverage.”
- Unstated goal: “Reduce test cycle time.”
- Goals specifically designed to be measurable.

Communication

Failed Project

- No consistent communication about project goals and status.
- Inadequate communication with executives.
- Inadequate communication with manual testers.

Successful Project

- Same detailed weekly status report sent to all. Status information available online at all times.
- Close communication with the VP of Development and Director of QA
- Verbal status reports delivered in weekly QA meeting.

Readiness to Automate

Failed Project

- Extremely limited test documentation; most testing ad hoc.
- No method of tracking test results.
- Testers lacked a strong understanding of how to test the product.

Successful Project

- Written test documentation. Each test case numbered individually.
- Test results tracked on spreadsheets that referenced the test case number.
- The test group as a whole had a much better understanding of how to test the product.

Role of the Automated Testing Team

Failed Project

- “Bulldozer builders vs. ditch diggers”
- Automators didn’t appreciate the value of manual testing.
- Manual testers felt threatened.

Successful Project

- Service organization focused on building tools.
- Automators understood that automation cannot replace manual testing.
- Manual testers more involved in the process and therefore less threatened.

Project-Level Differences Summary

- Experience level of leadership
- Goals of the projects
- Communication
- Readiness to automate
- Role of the automated testing team

Technical Differences

- Automated test system architecture
- Method of script creation
- Method of verification
- Programming practices

Automated Test System Architecture

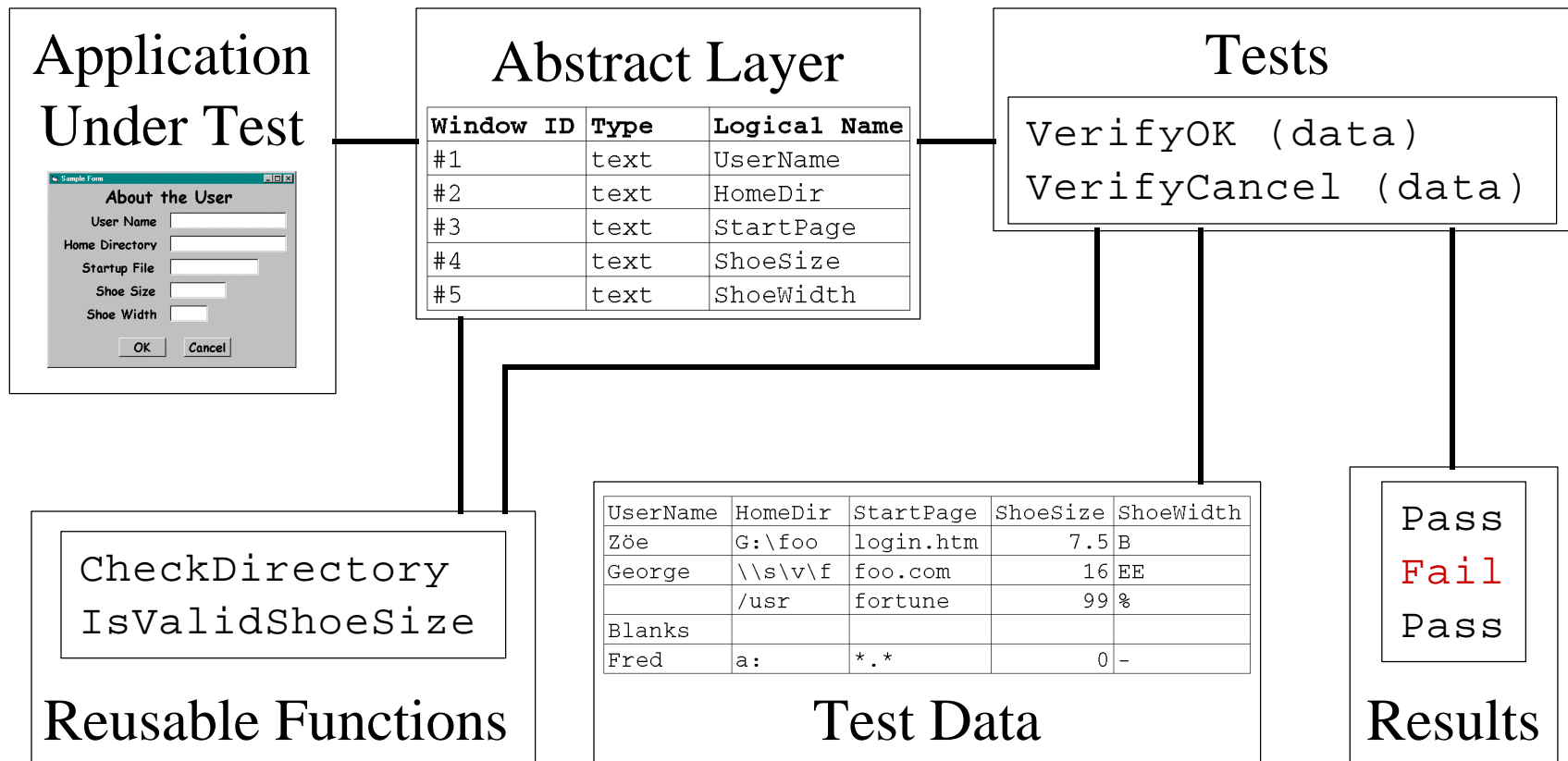
Failed Project

- Tool best-suited to creating individual scripts, not entire systems.
- Tool did not support creating a reusable library of functions
- No support for logical layer resulted in maintenance nightmare.

Successful Project

- Tool specifically designed to support creation of automation systems.
- Tool supported & encouraged creating a reusable library (“infrastructure”).
- Logical layer vastly improved portability & maintainability of scripts.

Elements of a Successful Architecture



Method of Script Creation

Failed Project

- Primarily record & playback
- No automatic test case or test data generation

Successful Project

- Primarily data driven; record & playback used as a learning tool only.
- Used advanced features in the automation tool to support automated test data generation.

Elements of Good Script Design

- Tests structured with setup, action, and result
- Tests are not order-dependent
- Test data is never hard coded
- Results are informative
- Pass/Fail determination is as automated as is practical

TestSomething

[Setup Actions]

Setup Actions drive the application under test to the point where the test can be performed.

[Test Action(s)]

Test Actions perform the test.

[Verify Results]

Verify Results evaluates the actual results against the expected results and determines whether the test passes or fails.

Method of Verification

Failed Project

- Bitmap comparisons to verify both window existence & contents of window.

Successful Project

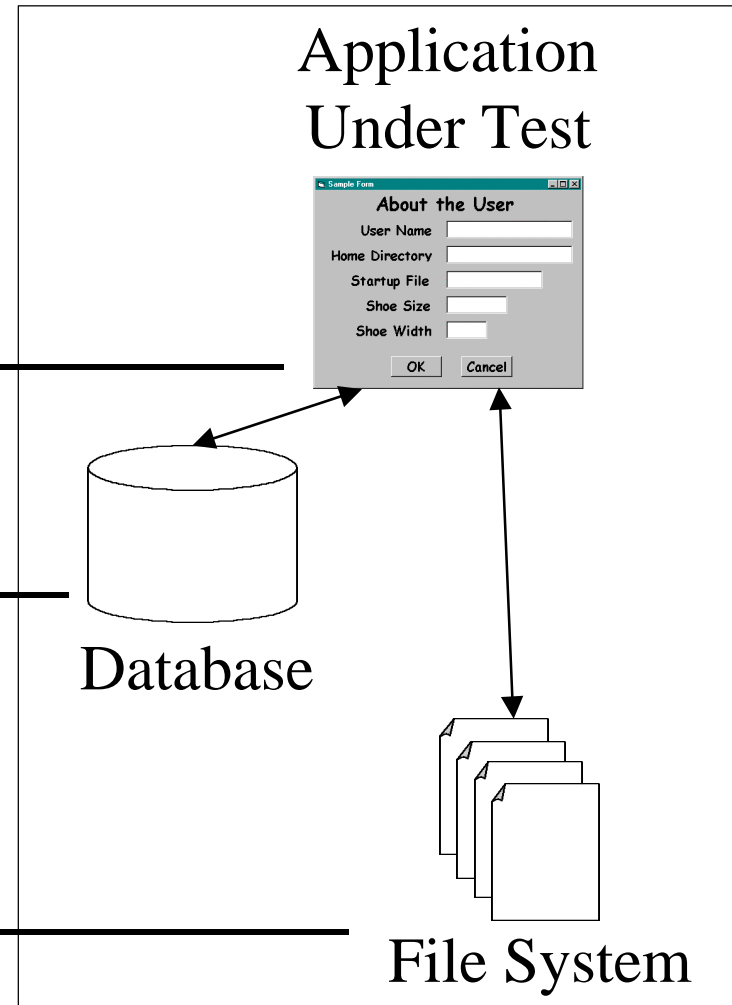
- Logical window existence functions to verify window appeared.
- Logical comparison between expected fields in window and actual fields.
- Test data verification

Techniques for Verifying Results

Use the application to verify displayed data, navigation, and window & object existence, appearance, and properties

Use calls to the database to verify actual stored data

Use calls to the file system to verify file existence, file attributes, and actual file contents



Automation Programming Practices

Failed Project

- Automation standards focused on file naming conventions.
- Extremely limited code reviews.
- No source control; test script management done exclusively through tool's management module.

Successful Project

- Automation standards focused on what constitutes good code
- Both formal & informal code reviews on a regular basis.
- Commercial source control system used.

Technical Differences Summary

- Automated test system architecture
- Method of script creation
- Method of verification
- Programming practices

What I Learned about Managing Automation

- Set realistic goals.
- Measure your progress toward those goals.
- Communicate goals and status clearly and consistently.
- Don't let your management set their expectations based on vendor hype.
- Coordinate with manual testers.

What I Learned about Creating Automation

- The right architecture can make everything else fall into place.
- Having the right tool for the job makes a difference.
- Simple scripts can be more powerful than complex do-everything scripts.
- Automation is programming: good programming practices apply.

Did the Successful Project Meet the Definition of Success?

After six months with 3 full time automators, including me:

- *Overall Cost Savings* - 50 hours/test cycle.
- *Improved Testing* - created hundreds of automation-only tests.
- *Shortened Software Development Cycle* - not yet...but more testing was being done in the same amount of time.
- *Reliable Results* - method of verification as reliable (if not more so) as visual verification.
- *Process in Place for Future Success*

The Difference between Test Automation Failure and Success

Elisabeth Hendrickson
Quality Tree Software, Inc.
esh@qualitytree.com

August 21, 1998

ABSTRACT

Automated testing is an expensive, resource-intensive activity. It can be difficult to achieve the return on investment on test automation that upper management expects. This paper examines two test automation projects in detail: one that succeeded and one that failed. Both projects were led by the same person at the same company and were intended to automate testing of the same software. The purpose of this paper is to illustrate the differences between the projects that led to success or failure.

INTRODUCTION

In 1993 I became the leader of a test automation effort for an independent software vendor. Six months later, the automated testing team was disbanded having made very little progress. The company initially attempted to integrate test automation with manual testing, but gave up all pretense of automation within two months. I moved on to other projects where I worked on automating my own tests sporadically.

In 1995 I began a serious effort at automating my current project, a Mac port of the main PC product line. My efforts were successful and became a pilot program for a larger automation effort. In 1996 I again was named leader of a test automation effort for the PC product line. The test automation project was much more successful the second time around. This paper examines the reasons for the first project's failure and the second project's success.

Project Similarities

Comparing these two projects is interesting because of their similarities: same company, same product, and same leader.

The company is a medium-sized independent software vendor. I led both projects. The product is a complex GUI-based client/server application consisting of many different executables for performing different tasks. The product has an open architecture, supporting a variety of databases (Oracle, Sybase, MS SQL) and network operating systems.

Project Differences

Although the projects occurred in similar environments, the projects themselves were vastly different. The differences fall into two categories: project-level differences and technical differences. Neither category of differences should be discounted; both contributed heavily to the failure of the first project and success of the second.

Project-level differences include the experience level of management, the goals of the project, and the organization's readiness to automate. Technical differences include automation architecture, methods of verification, reusability and maintainability of test code, and programming practices.

PROJECT-LEVEL DIFFERENCES

This section examines the project-level differences:

- Experience level of leadership
- Goals of the projects
- Communication
- Readiness to automate
- Role of the automated testing team

The First Project: A Case Study in How Not to Automate

The project that failed is a case study in how not to automate. Unfortunately, perhaps because of the tendency of the automated testing tool vendors to oversell the ease of use of their products, the mistakes we made are all too common.

The problems that led to failure included inexperienced leadership, unrealistic goals, and the fact that the organization was unready to automate.

Inexperienced Leadership

When I accepted the role of team lead, I was inexperienced both in leadership and in test automation. The QA manager also lacked experience with test automation projects. This became a critical weakness as pressure for a return on the automation investment mounted. The more pressure came to bear, the more morale dropped in the automation group, and the less the automated test group achieved.

Because of my inexperience, I had difficulty communicating the technical problems hampering the team to the QA manager in a way that he could understand. I also had difficulty communicating the business objectives of automation to my team. Finally, I spent far too much time creating Gantt charts of a fictional schedule and far too little time automating myself.

Unrealistic Goals

The goal that I was given when I started was "Automate Everything." I should have questioned this goal. When it became apparent that everything could not be automated in 3 months (the original time frame for the project), the goal became "Bang for the Buck." We didn't succeed at defining effective "bang for the buck" activities. Thus, the goals were nebulous and we had no means of measuring our progress against the goals.

Communication with Upper Management

Inexperienced leadership led to lack of communication with upper management about how their money was being spent and what benefits they were gaining from tests automation.

Consequently, there was greater pressure to reduce investment in automation because payoff was not obvious.

Unready to Automate

Finally, even if we had been able to resolve all of the above problems, there was still one major obstacle: the organization barely understood how to test the product in a manual way. The tests needed to be designed before they could be automated.

Role of the Automated Testing Team

We in the automated testing team did not see ourselves as a service organization. We saw ourselves as tool builders. We intended to build the tools for which we saw a need without conferring with our “customers” (the manual testers).

The automation team also suffered from lack of respect for manual testers. We viewed manual testing as drudgery not to be endured.

To this day, I am deeply embarrassed when I think about the analogy I drew between automated testing and construction. Upper management wanted to pull the automation team members off automation to test a patch release manually. In fighting this attempt to derail test automation, I used the following analogy: “There's a group of people digging a ditch with shovels. We're building a bulldozer. You want us to drop what we're doing and pick up shovels.”

I'm deeply embarrassed because:

1. This analogy conveys the idea that I think manual testers are akin to ditch diggers. At the time, I was sufficiently naive that I might actually have thought that. I have come to my senses. I realize what an offensive statement this was.
2. I have come to realize that sometimes it really can take too long to build a bulldozer.

Imagine a forest fire. Fire fighters are working hard to build a firebreak by digging ditches. The bulldozer has just arrived, some assembly required. A group of people breaks off from the fire fighting effort to assemble the bulldozer. They discover that it will take them much longer than anticipated to assemble the bulldozer and that they do not have the right tools for the job. They can either wrestle with the bulldozer or go back to fighting the fire.

If the fire is sufficiently bad and there are too few firefighters, it may be in the best interest of the organization if the firefighters go back to digging the fire break and give up on the bulldozer for now. After the fire is under control, the would-be bulldozer assemblers can see about finding the right tools and spending time on assembly.

If the organization suffers from one fire after another and the test group is always in fire fighting mode, automation will not help and might hurt. In crisis-driven organizations, it is important to look at the root cause of the fires: process, management, goals, and methods. After some of the root cause factors are addressed, tools such as automation can be brought in.

The analogy also suggests another possible issue with assembling the bulldozer: the bulldozer may not be able to operate on the terrain. Similarly, not all test tools will work well with all applications.

Automation is not a silver bullet by any stretch of the imagination; it is a tool. As with any other kind of tool, it can be very effective if used well, but it's also very easy to cause severe damage to yourself, the project, or the organization by using it unwisely.

The Second Project: Key Factors in Success

A few of the most crucial factors in the success of the second project mirror the critical factors in the failure of the first. The second project had experienced leadership and realistic goals, and by the time the second project got underway, the organization was ready to automate.

Experienced Leadership

The entire management chain, all the way up through the president of the company, changed between the first project and the second. The new QA manager had been through a number of test automation projects and understood the challenges inherent in test automation. I had two years additional leadership and automation experience under my belt. Finally, I was able to draw on my experiences in the first project to steer the second project away from previous mistakes.

Realistic Goals

In the first project, I was handed the project goals. In the second project, the automated testing team defined the goals we believe to be most appropriate and presented them to management. The goals of the second project were much narrower and more measurable:

1. Save the manual testers time (measured in hours)
2. Improve test coverage (measured in tests that could not be run manually)

The agenda behind the project goals also changed. In the first project, the focus was on “Automate Everything” in part to reduce the number of manual testers needed. In the second project, the focus was on supporting the manual testers and shortening the testing cycle.

It is understandable that the manual testers did not support the first project while they did support the second project. They felt threatened by the first automation initiative (as well they should have since one of the goals was to reduce the need for manual testers). Internal departmental support is critical.

Not all levels of management agreed that these were the right goals for the project. However, because the QA manager agreed with the automated test team’s goals, we were able to provide a united front to upper management. This enabled the project to stay on course and kept individuals on the project from being distracted by “special requests.”

Communication

With the shortcomings of the first project firmly in mind, I made sure to communicate goals, successes, and stumbling blocks clearly to the automated testing team, the manual testers, and all levels of management that showed the faintest interest.

The automated test team produced a comprehensive weekly status report that included measures of our progress toward our goals as well as current projects, accomplishments, problems, and future plans. This may seem like overkill, but it only took me about 2 hours a week to put this information together from individual automator’s status reports. Creating this report kept me from having to answer the same questions about our status repeatedly.

Distributing the same information to all interested parties had the additional benefit of ensuring that everyone had the same understanding of the project goals and status; this reduced the number of conflicting demands.

Ready to Automate

Between the first automation project and the second, the QA organization matured substantially. We had created real test cases and test plans, established methods of tracking testing progress,

and had a much better understanding of how to test the product. This understanding enabled the automated testing team to focus our automated testing on the important functionality. It also enabled us to better support the manual testers.

Role of the Automated Testing Team

In the second project, the automated testing team became a service organization with the manual testers as our customers. Rather than attempting to design the automated tests in a vacuum, we worked with the manual testers to make sure that the scripts we created had value for them.

In addition, testers outside the automated testing group were encouraged to learn the automation tools. We held classes and worked with testers one-on-one to help them learn how to automate their own tests and setup activities. There were three advantages to doing this:

1. The testers gained a better understanding of what we were trying to accomplish.
2. Many of the testers gained confidence in their ability to contribute to automation.
3. The testers understood that they were a big part of the process, not an inconvenience to be replaced by automation.

The ultimate vision of the automated testing group was that we would be a small core group of automators who would create and maintain the automation infrastructure (harness). The manual testers who had an interest in automation would automate their own tests. Manual testers who did not have an interest in test automation would not be forced to automate: we recognized that not everything could be automated and that there would always be a need for purely manual testers.

TECHNICAL IMPLEMENTATION DIFFERENCES

Technical decisions played a large role in the failure of the first project and the success of the second project. Some key technical differences between the first project and the second are:

- Automated test system architecture
- Method of script creation
- Method of verification
- Programming practices

The First Project: Non-Reusable, Non-Maintainable, Prone to False Failures

Automated Test System Architecture

By “architecture,” I mean overall design and infrastructure. On the first project, we were unable to create a real architecture in large part because the tool did not provide a good way to organize scripts or to build reusable libraries of functions.

Method of Script Creation

The primary method of script creation in the first project was record & playback. We did modify the recorded scripts to add control of flow statements (if, while, for, etc.), but we did not write scripts from scratch.

Method of Verification

Perhaps the biggest mistake we made in the first project was focusing on bitmap comparisons for verification. The tool supported other methods of verification; we mistakenly thought that bitmap comparisons would allow us to verify the UI more completely. There are a number of problems with bitmap compares: they produce frequent false failures, they are difficult to manage, and they take up a lot of disk space.

Bitmap compares produce frequent false failures because simple, innocuous differences can make a bitmap compare fail. For example, running the script on a machine with a different resolution or color setup and cosmetic UI changes like reformatting a message caused failures.

Programming Practices

Because we used record & playback to create scripts in the first project, our programming standards focused more on file naming conventions than good programming practices. We never held a code review, we did not use source control, and we made basic programming gaffes like hard coding values into our scripts. In addition, the tool did not support creating a reusable library of code, so after an initial attempt to create a library of functions, we gave up. We spent a large amount of time duplicating effort and maintaining redundant code.

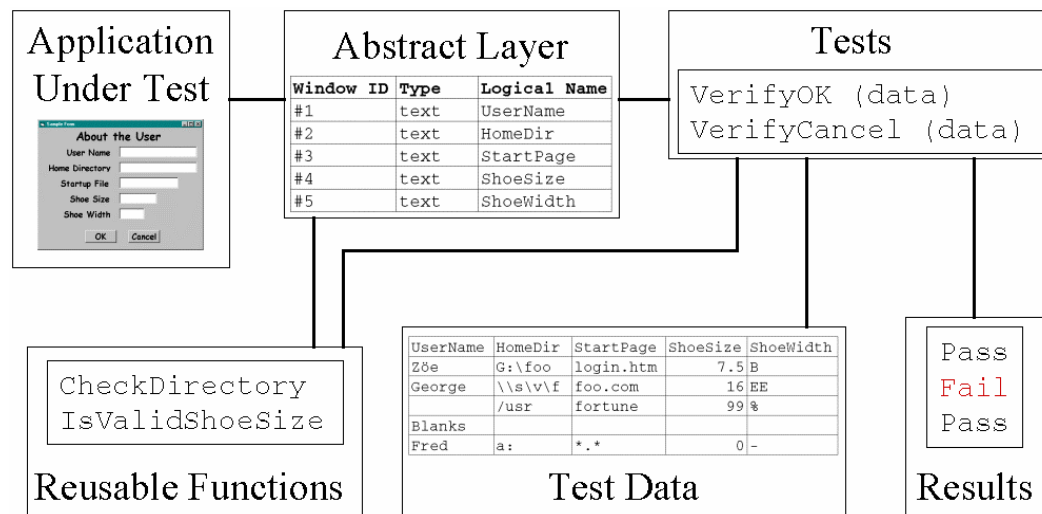
The Second Project: Test Automation is Programming

The most important lesson I learned on the first project is that test automation is programming and must be managed accordingly. Record & playback does not result in robust automated tests.

It's important to design the test system up front, to write specifications, to document your work, to use source control, and to do all those other things that we Quality Assurance types keep telling Development they should be doing. The essential factors in the success of the technical side of the second project illustrate this.

Automated Test System Architecture

In the second project, we were able to define an overall architecture for the project. We chose a tool that supported reusable functions, data driven testing, and logical mapping between program elements and the scripts.



This paper does not focus on the tools used, but rather on the way in which we used these tools. That said, it is important to note that between the time the first project started in 1993 and the second project started in 1996, the testing tool industry had matured significantly. We had many more tools available to us and the tools were more flexible and powerful.

ABSTRACT LAYER

This abstract layer, called a GUI map or Test Frame depending on your system, made it easy for us to adapt to product changes.

We were able to create application element declarations to provide a logical layer between the application under test and the test scripts. We spent half our time or more creating these declarations and associated reusable functions.

To some, this might have seemed like wasted time since it produced no executable scripts. However we more than made up for the time investment in maintainability. When the interface changed in mid-release, it was a simple matter to change the application element declarations in one place. In the first automation project, where there was no logical layer between the application and the scripts, a simple interface change could result in hours of rewrites and fixes to make existing scripts work.

REUSABLE FUNCTIONS

Where the first system did not support reusable functions, the second did. We were able to use reusable functions to reduce the amount of code required for each script. In fact, once the infrastructure was in place, creating new test scripts was trivial.

TEST DATA

With the second automation project, we were able to realize the power of data driven testing. Instead of creating a hundred scripts, we could create one script and a data file with a rows of data.

Method of Script Creation

In the second project, we used Record & Playback only as a learning tool. Most of the automation team was writing the vast majority of their scripts by hand within a couple of months after the start of the project.

Writing scripts by hand takes less time and is less error prone than record & playback, once you know the automated test tool. In addition, writing scripts by hand requires that you think through how the script will work ahead of time. Up front design is as important in automated testing as in programming.

Method of Verification

In the second project, we moved away from bitmap compares and used logical window existence functions. This greatly reduced the false failure rate and sped up script development and maintenance. In other words, rather than testing to see if a window that looked exactly like the picture appeared, the scripts checked to see if a window with the right name appeared. This technique gave us more useful information than a bitmap compare failure. For example, we created a smoke test that reconciled the fields expected to be on the form with the fields that actually appear.

Programming Practices

In the second project, we followed the sort of programming practices that we advocate for developers: coding standards, source control, and code reviews.

Coding standards made it possible for all of us on the automation team to share code: everyone could understand everyone else's scripts and functions. Our coding standards that covered function and variable naming conventions, hard coded values (don't), commenting, good test script design, etc. It's important to note that we spent a lot less time defining the standards on the second project even though the second project's standards were more comprehensive and ultimately more useful.

Source control saved us from rework. We put our libraries and scripts in source control almost from the very beginning on the second project. This proved to be invaluable. If we lost changes by copying over a file, we were able to get back the changes immediately: we did not have to search for the file on a backup tape or redo the work. In addition, source control gave us a list of changes for activity tracking purposes: as the manager, it was easy for me to tell who was working on what by looking at who had which files checked out.

Code reviews allowed us to share techniques, verify adherence to standards, and improve reliability. During my time with the project, we held two formal code reviews and innumerable informal reviews. We did not even attempt code reviews on the first project.

CONCLUSION

The following table summarizes the differences between the successful project and the failed project:

	On the failed project:	On the successful project:
Management Issues	Upper management had unrealistic expectations set primarily by the test tool vendor.	Upper management had realistic expectations set by the project leader.
Project Leader	Inexperienced	Experienced
Project Goals	The project goal handed to the automation team was: "Automate Everything."	The project goals that the automation team agreed on were: "Save manual testers time; improve test coverage." Progress towards these goals was measured and the results reported to management as well as the manual testers.
Technical Issues	We did not use source control and occasionally lost work as a result.	Real source control; better coding standards; tool supported error recovery and building reusable libraries of functions.
Type of Automation	Primarily record & playback	Primarily data driven, almost no record & playback

Lessons Learned

I learned a number of lessons from participating in both projects. Specifically, it is important to:

1. Set realistic goals and find ways to measure progress toward those goals.
2. Hold off on automation if the organization is not ready for it.

3. Set management expectations appropriately.
4. Coordinate automation efforts with the manual testers to make sure there is no duplication of effort.
5. Communicate openly and often about the project goals and status with everyone involved including the test automation team, the manual testers, and all levels of management.
6. Look for simple scripts that have large payoff.
7. Treat test automation like program development: design the architecture, use source control, and look for ways to create reusable functions.
8. Focus on creating robust, reusable, maintainable code, not on banging out test scripts.
9. Use the right tool for the job.

Elisabeth Hendrickson

Elisabeth worked for a variety of companies in the software industry for over a decade before founding Quality Tree Software, Inc. (formerly Quality Tree Consulting) in 1997. As a consultant, Elisabeth helps her clients define and execute test strategies, implement automated testing, establish quality criteria, and improve overall product quality. Elisabeth has first-hand knowledge of all phases of software development, having at one time or another been a programmer, tester, technical writer, support technician, and manager at industry leaders like PC DOCS and Sybase. Elisabeth is a regular participant in LAWST (the Los Altos Workshop on Software Testing) and a frequent speaker at conferences and local user groups.

Elisabeth Hendrickson
esh@qualitytree.com
Quality Tree Consulting

<http://www.qualitytree.com>
voice: 925-560-0530
fax: 925-560-0531