Michael Prisby

Michael Prisby has 10 years of experience in the Information Technology realm specializing in Quality Assurance and Test Automation. He has a Master of Science degree in Computer Information Systems, and is the current Associate Department Chair for the Information Technology Department at the University of Phoenix, Maryland campus. His current position is the Test Automation Lead for the Customer Automation and Internet Systems department of United Parcel Service (UPS). In this capacity he has masterminded the automation of three website applications, used in over 36 countries in 11 languages and 19 different dialects, that can be viewed at www.ups.com.

Beyond Record and Playback:
A Behind-the-Scenes View of Web Test Automation

By Michael Prisby
Test Automation Lead
Customer Automation – Internet Systems
United Parcel Service (UPS)

Preface

"META Group research indicates that only 20% of companies test Internet projects before putting them online—this despite research on e-commerce site usability that found almost half of the test population had trouble making purchases. The test shoppers completed only 56% of the shopping tasks that researcher assigned to them." (**www.metagroup.com** (31 December 2001)).

In the Quality Assurance (QA) realm, we have all heard the phrase, ***"do more with less."*** This catchy little saying has been floating in our midst for years. Unfortunately, for many QA departments trying to ***"do more with less"*** actually becomes ***"do less with less."*** The methodology of Planning, Analysis, Design, Develop, Implement, and Support or PADDIS is a proven concept in test automation. It's based upon a blend of Project Planning, Software Development, and Quality Assurance. These three components, when applied to the automation of website testing, form the baseline for success. PADDIS is not based on theory; it's based on actual, hands-on, successful website automation projects. The Planning and Analysis portion addresses issues such as automated test tool selection, evaluating your test team's ability to use and understand the automation assets, and what to automate. Design and Development will help you understand two very important concepts: why automation is not automatic, and why test automation is actually a development effort rather than a pure Quality Assurance effort. Finally, Implementation and Support will assist you in understanding what needs to be done to get the automation off the ground and how to maintain it once its there.

Planning and Analysis

Planning and analysis go hand in hand. You cannot possibly achieve a successful automation plan without adequate and thorough analysis of that plan. These first two steps in the PADDIS methodology are, and will forever be, the most important phase(s) of test automation. The need to provide you and your test team adequate time to plan and analyze is imperative.

The test automation effort should be looked upon just like any other project; there should be no difference in your approach to planning and analyzing. Planning, and the analysis of the plan, have always answered the core questions of project feasibility; Why, What, Who, Where, When, and How.

Why?

'Why' are we automating? You must be prepared to answer this question up front not only for yourself and your test team, but for management as well. Automation is not automatic. It will take time, money, and dedication by all parties concerned for its successful implementation. The automation of website testing may not help you to reduce your Quality Assurance (QA) staff, nor will it drastically reduce your time to test. So, the question is proposed again, 'Why' are we automating?

The answer to the 'Why automate?' question can be summed up in one simple word: Efficiency. It has always been the responsibility of the QA department to ensure that the Application Under Test (AUT) is free of all defects and that it meets all of the functional requirements. However, as AUT's mature, new functionality is added, requiring more testing. The problem with this scenario is within time; the QA department will fall behind and will not be able to keep up without requiring additional resources. Unlike Development which only deals with current functionality and its dependencies; QA must prepare for and test both current and past functionality. In the eyes of QA personnel, functionality is regularly added but seldom taken away, therefore, testing just keeps growing and growing. Instead of "doing more with less", it actually becomes "doing less with less".

QA departments are constantly asked to "do more with less". Relief only comes when they are at the breaking point. Because of this QA's efficiency is affected; requiring QA to cut corners to meet testing deadlines. There are two solutions to this dilemma: add QA resources or become more efficient by automating. The first option is a never-ending fix, meaning you'll always have to add QA resources to keep afloat (continually add QA resources? - like that's going to happen!). The second alternative is a more viable and lasting solution. By automating, not only will your testing become more efficient and effective, but you'll actually be able to catch up and stay abreast with the Development department. Now, it's feasible to "do more with less".

What?

The next core question that we must ask ourselves is two-fold. What are we going to automate and what test tool are we going to use to do the automation (if any)? You must know the answer to the first question before you can answer the second.

What to automate?

We want to automate everything! A nice thought, but the reality is that an AUT comprises the integration of many tasks. There will always be some type of manual processes involved and for that reason, the goal of becoming 100% automated may not be feasible. So, if we can't automate everything what can we automate? Ideal candidates for automation of your website testing are those that are repetitive in nature. Meaning, if you have test cases that go from point A to point B multiple times, and the only thing that changes is the test data, then that type of test case would be a perfect candidate to automate. Additional examples are regression scripts or global/internal links tests. Even though the last two examples are tests that normally run once, they still fall into the category of repetitiveness because they're executed on a regular basis, every release, for instance.

Now that we have the answer to our first 'What' question, we can move forward to determine what test tool to use. You must use caution when researching automation tools. We've all heard the horror stories about QA Departments spending thousands of dollars purchasing test automation tools and ending up with expensive pieces of shelfware. You hear comments like; "Their software didn't work", "The vendor lied or misrepresented what their product does". However, it is not always the vendor's fault. If you haven't answered the first 'What' question thoroughly, you might not accurately state your needs to the vendor. Unfortunately, this problem is normally not discovered until after you've purchased and started using the test automation tool. Fortunately, there is a way to avoid this pitfall, it's called a "Trial Version." Many vendors will provide you with a 30 to 90 day trial version of their test tool application for free or for a very low fee. In some cases, the vendor might even volunteer to come to your site and give you and your team a demonstration. Take advantage of this opportunity because your selection of the correct test tool will determine your success or failure of your test automation project.

Who?

'Who' must address the assets question. 'Who' will be necessary to make test automation successful? The process of test automation is closely likened to the process that development performs to initially create an application. You might be wondering why I'm addressing this question in the 'Who' category? It's simple; in order for you to have success at test automation, you must get out of the mind set that this is a purely a QA or testing effort. It is not! Add the word 'development' to the end of 'test automation' and you'll start to see that 'test automation' is actually a development effort rather than a QA effort.

Test Automation Development

Almost every automation tool on the market has some type of record and playback feature for website testing. This is a given. However, to go beyond mere record and playback and truly automate your application, your test team will need to manipulate, add functionality, and input verification points within the recorded scripts. The challenge is this, the recorded scripts are written using some form of programming language (C, C++, Visual Basic, VBA, or vendor unique language – just to name a few). In order for your test team to manipulate these scripts,

they'll need some knowledge and experience in programming concepts. 'Who' on your test team has such skills ('Who' should I get if the answer is none?). Should you look to outsource or maybe find and request someone inside your organization – one of the developers for instance? Ideally, you want your automation test team to consist of test personnel, programmer(s), and if possible analysts (hopefully, your testers already have good analytical skills then the analysts won't be necessary).

Where?

'Where' is another two-folded question. 'Where' are we going to do the test automation (in-house or outsource), and if in-house, 'Where' are we going to get the resources (hardware, space, etc.) to achieve our automation goals?

'Where' are we going to do the test automation (in-house or outsource)?

It's already been proven, in Management 101, that in-house development (in most circumstances) is preferred over outsourcing. In-house development is normally cheaper than outsourcing, it helps to maintain control and ownership of the assets created, and it takes away the dependency upon outside sources. However, what happens when you are answering the 'Who' question and realize that you don't have the human resources required to complete the test automation effort? Do you scrap the automation idea altogether? Why not outsource, is outsourcing really that bad? The answer to these questions depends primarily on your need to automate. If the need to automate is just a simple want or a nice-to-have, then you'll probably stop after getting a negative answer to the 'Who' question. However, if automation is a requirement and if not done, would hamper your ability to adequately test your application, outsourcing might be your only alternative.

Outsourcing

Outsourcing needs to be approached as cautiously as the automation tool procurement. You must know your needs and wants prior to soliciting for proposals from outsource companies. Remember, the successful creation of your automation testing will depend greatly on your ability to relay your needs and wants to the outsource company. There are plenty of positives that can result from outsourcing. The biggest of which is, your web application will be automated. Additionally, outsourcing personnel can provide you and your test team a strong foundation upon which to build. It can also provide you a blue print for all future automation efforts.

'Where' are we going to get the resources (hardware, space, etc.) to achieve our automation goals?

Enough about outsourcing. Let's assume that when you answered the 'Who' question you determined that there's enough personnel in-house to achieve the test automation goals. Now we must figure out where to procure, borrow, or swap the hardware, software, and possibly lab space that will be needed to complete the test automation project.

The only way that you can accurately determine where you're going to actually develop, and execute your test automation scripts is to narrow the overall choices of the automation tools. At this point in the process it's easier to determine what you need if you have less overall options from which to choose. All test automation tools have specific hardware and software requirements and configuration issues that must be met and or addressed in order to receive optimal performance within the automation tool. Issues such as hard-drive size, CPU speed, RAM, operating system, and software compatibility must be thoroughly investigated to ensure that you have the resources to develop, execute, and store the automation project.

When?

The 'When' is a very tricky question. It's easy to get engulfed by the automation whirlwind and get overly excited about all of the possibilities. My only advice in this manner is to follow the PADDIS methodology and take your time. If you listen to most automation tool vendors, you might make the mistake of under estimating the completion time. Additionally, pressure from management, expecting immediate results can contribute to the under estimation of project completion.

The actual completion time will depend greatly on the following factors:

- Resources (human and non-human),
- The maturity of your current manual testing process, and most importantly,
- The initial plan and the thoroughness of the analysis process.

Another factor that plays a significant role in timeline predication is automation tool familiarity. Additionally, the day to day commitments of a QA team can also interfere with the automation project timeline. These are just some of the many items that must be accounted for, when predicating timelines. These can be greatly minimized if not totally nullified if you and your test team take advantage of the 'Trial Version' of the automation tool, have a mature manual test process currently in place, and adequate availability of human and non-human assets. Remember this; there is no golden timeline for completion of your automation project. In my experiences, I have seen some automation efforts span over a year, while others were completed in less than 30 days.

How?

After all of the previous core questions have been answered, the final question is 'How?' are you going to put all these wheels in motion? First, you must be able to convince upper management (or the check signer – whichever you prefer) 'How?' automation would be beneficial to the company. This is easier to accomplish if you have thoroughly answered the previous core questions. Secondly, all parties concerned must ask themselves 'How?' feasible is this concept and/or plan. Can you be successful in this endeavor? Lastly, you must know 'How?' to transition from Planning and Analysis to Design and Development.

Design and Development

Visualize your website as a completed puzzle. During the design phase, the goal is to slowly pull the puzzle apart, piece by piece, and put the individual pieces back into the box. The

difference between this and a real puzzle is as you are putting the pieces back in the box, you are labeling and categorizing them to be referenced and used later during the Development phase.

The PADDIS Design and Development methodology is based on the use of data driven testing concepts. The PADDIS methodology can be used by other test automation design and development concepts (e.g. key-word driven), but the data driven concept is what I have personally used time and time again to automate applications on our UPS.com website.

Design

During the design phase, a total dissection of the current manual process is performed. This phase could be likened to the concept of reverse engineering. To design test automation, one must go well beyond the simplicity of analyzing, and possibly recording, the manual tests scripts. One must return to the root of all testing, the functional requirements and the business rules. Remember that manual test scripts are nothing more than one person's interpretation of the functional requirement, the related business rule, and the 'as-built' system (for legacy applications). When you automate, these same interpretations will change. The design phase primary goal is to start breaking up the whole into small, common requirements or modules by using a Top-down design approach.

In a normal development lifecycle, during the design phase, items such as technical specifications and use cases are created. Depending on the maturity and the refinement of your manual testing, QA could have the equivalent to these items within their current manual test cases or scripts. What is a mature and refined test process, you ask? In order to consider your manual test process as mature and refined, your manual test scripts must include, at a minimum, these following things:

- Adequate test script tractability to the functional requirements and/or business rules.
- A Step by Step approach used within the test script – No assumptions should be made. Remember, once you're in the development phase there may be personnel working on this effort that know little or nothing about your AUT.
- Very specific GUI and data-related verification points within the test script.
- Knowledgeable QA personnel with regards to the Functional requirements and Business Rules.
- Test Documentation.

If your manual test process has not met the criteria stated above, you should seriously consider starting to prepare Use cases. Use cases could be considered abbreviated versions of test scripts because Use cases focus on very specific pieces of the total functionality rather than combing functionality like test scripts do. It is not uncommon to have a 1 to 1 relationship between Use cases and Functional requirements and/or Business Rules. Depending on the size of the AUT and/or the amount of Functional requirements and/or Business rules, creating Use cases could become very time consuming affecting the overall timeline of your automation project.

The creation of a Physical model to depict what the website does as well as the flow(s)/path(s) within the website is very useful in the Development process. A simple Data Flow Diagram

(DFD) showing all screen shots and the flows/paths is priceless in the areas of project timeline and overall understanding of the AUT.

Additional issues such as, coding standards, peer reviews, naming conventions, configuration management, and the high-level structure of the development process must be addressed and put in place during the design phase in order to properly and successfully transition to the development phase. These don't necessarily have to be completed before moving forward; however, you don't want to wait until after you're in the development phase to start addressing them.

Development

Developing automation scripts is very similar to developing the actual AUT. Via the design phase, automation development begins by taking the broken down common requirements and/or modules and turning them into very specific functions or actions.

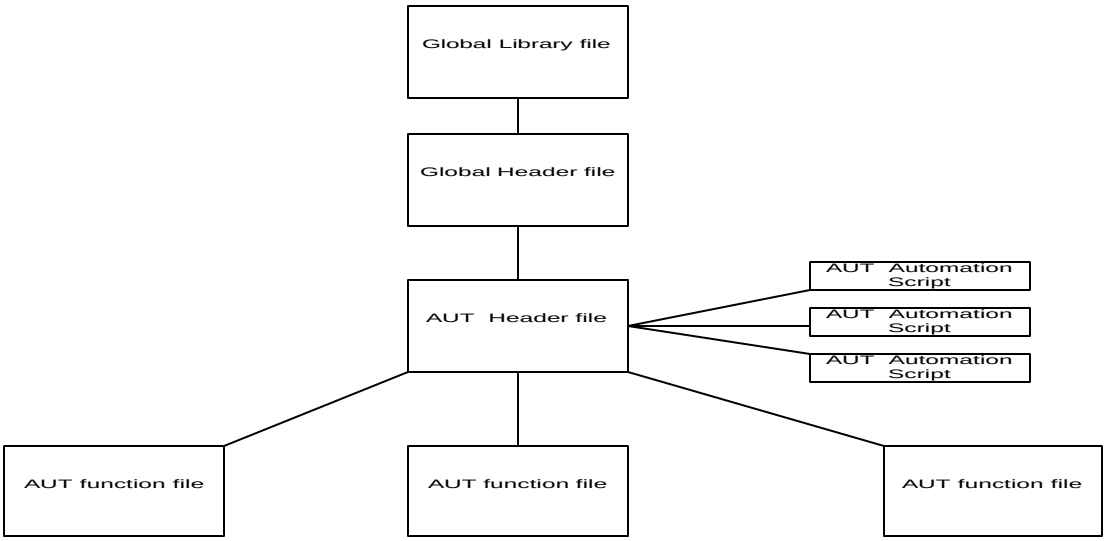
The Development phase is comprised of many small tasks. The issues of coding standards, peer reviews, naming conventions, configuration management, and the overall structure of the code must be addressed and tentatively finalized early in this phase. This is only the beginning. The actual creation of the automation scripts is the core of the Development phase. In this section, we will address the Automation Development Architecture, Record and Playback Scripts, and Automated Script Decomposition.

Automation Development Architecture

The Automation Development Architecture was primarily developed to reduce maintenance and duplication of functionality for automated testing. However, after using the PADDIS methodology to automate two web applications, it was discovered that some of the manual test scripts performed very similar, if not the exact same, functional pieces. Both AUTs, for example, used the Login function. By adding a Global hierarchy, the Login function was elevated to a Global function, therefore allowing any AUT that includes the Global Header in their automated scripts the use of the Login function.

A description of the components of the Automation Development Model (seen below) is as follows:

- Global Library – Stores the Global functions (i.e. Login function)
- Global Header – Stores the Global functions declarations (i.e. Points to the Login function)
- AUT Header – Stores the AUT function declarations (i.e. Points to the AUT specific functions)
- AUT Function (Library) File – Stores individual AUT functions
- AUT Automation Script – The actual automated script executable (i.e. TestCase001)



Automation Development Model

Record and Playback

Simple record and playback does have some useful benefits. Frankly, if it weren't for this feature, website test automation would be extremely difficult. Record and Playback gives you and your test automation development team, a solid foundation to work from. As you are recording the manual test script, the automation test tool is writing most of the code for you. In addition, it helps in determining the different data and path flows, once the script is played back.

Hint: When using the record feature of your test automation tool, attempt to record your web application using the record by object name rather than record by coordinates. You want to do this because object names change less frequently than the coordinates do. The record by coordinates method is GUI and position-based, so, if the GUI changes so will the coordinates – this becomes a maintenance nightmare.

Don't get carried away with recording and playing back the manual test scripts. There isn't a need to necessarily have a 1 to 1 relationship between the manual test scripts and the automated test scripts. The majority of test scripts have common functionality and flows/paths. They normally differ in the expected result or where the test scripts end. For instance, if your AUT requires you to log in to gain access to the site or webpage then it is possible that every one of your manual test scripts starts with this step. However, for automation purposes, the log in process only needs to be recorded once. The log in script will become part of the global function library.

Hint: Once you've recorded a piece of the AUT functionality there is no need to record it again.

Once the manual test scripts and Use cases are recorded, the actual development process begins. This is the point where the script developers step in and start decomposing the recorded scripts and start creating individual functions.

Hint: Functions are very specific processes or actions.

Example:

- Login (Manual Use case):
  Step1 – Insert UserID
  Step2 – Insert Password
  Step3 – Click Submit button

- Login (Automated):
  Step1 – Call function Login (Login function = Step1 + Step2 + Step3 of the Manual Use case)

To log into the AUT, the user must supply the User ID, Password and click the Submit button. Within test automation these three separate actions are combined to become one single action called the login function. Combining specific manual test steps and creating functions will greatly reduce duplication and the maintenance of the automated scripts.

Automated Script Decomposition

The decomposition of the automated scripts is identical to the dissecting of the manual test scripts in the Design phase. The script developers break down the recorded code to form simple functions (puzzle pieces). The individual functions are then placed into Library files. The AUT Library files become the puzzle pieces and the AUT Automated Test Scripts grab the different puzzle pieces and puts them together to form multiple puzzles (i.e., automated test scripts). This process is described in the following steps.

Step 1 – Group multiple manual test steps into very specific and generic functions. Using the Login example from above, the manual test steps for Login are combined into one single function. To break this function down to its simplest level, making it generic, the 'Click Submit button' would be taken out and it would become its own function. Remember that we want to treat script development like a software development effort. And for that reason, you must think about the reusability of the functions you create. The function 'Click Submit button', could be reused within a different automated test script.

Example:

- Registration (Manual Use Case)

Step1 – Insert Name
Step2 – Insert Address
Step3 – Insert State
Step4 – Click Submit button

- Login (Manual Use case):

Step1 – Insert UserID
Step2 – Insert Password
Step3 – Click Submit button

- Registration (Automated):

Step1 – Call function Registration (Registration function = Step1 + Step2 + Step3 of the Manual Use case)
Step2 – Call function ClickSubmit

- Login (Automated):

Step1 – Call function Login (Login function = Step1 + Step2 of the Manual Use case)
Step2 – Call function ClickSubmit

- ClickSubmit (Automated):

Step1 – Call function ClickSubmit (ClickSubmit function = Step3 & Step4 of the Manual Use case)

The Registration and the Login functions can use the ClickSubmit function simultaneously. Additionally, it gives the automated test script creator more flexibility in determining the flow of the test script and it can reduce the amount of overall script maintenance. This will be explained in more detail in Step 3 – Create AUT test script.

Step 2 – In this step, the recorded functions are enhanced and placed into individual AUT Libraries. The degree of enhancement will have a profound affect on the amount of maintenance required to keep the scripts current and running.

Test automation development can be classified into three levels. All three levels vary in the degree of enhancements, but they all have their own strengths and weaknesses.

Level	Type	Strengths	Weaknesses
1	Simple record and playback	Good for ad hoc, defect producing/verifying, and run once type testing	High maintenance, lack of reusability, very specific, no flexibility
2	Replacement of hard-coded values	Begins data-driven technique, data inputs can be automated, more flexibility	High maintenance and moderate flexibility
3	PADDIS	Very low maintenance, high flexibility, and optimum reusability	Creation and maintenance depends upon programmers or QA personnel programming skills

Table 1 – Levels of Test Automation Development

To get to Level 3 automation, using the PADDIS methodology, additional tasks must be accomplished in Step 2. These tasks are listed below.

- Remove all hard-coded data and replace with variable names – Level 2 and Level 3
- Remove dependency upon coordinates, replace them with object names if possible (refer to previous hint).
- Remove and create separate button click functions
- Store functions in individual Library files
- Create Global Header/Library (if needed)
- Create AUT Header file

Step 3 – If Step 2 is completed to Level 3 standards, than it is time to create the automated test scripts. Creating automated test scripts is a team effort. The script developers need the QA group to assist them with determining the flow/paths that the test scripts must take (as described in the DFD). The QA group needs the script developers to explain and document what the different

functions do and how to use them. The automated test scripts are considered the completed puzzle. However, each puzzle (script) is distinctly different than the other. The puzzle is created with a very specific purpose. Even though some of the test script flows/paths may start off the same, their overall purpose and ending point can differ greatly.

Hint: You do not have to have a 1:1 relationship between your manual test scripts and automated test scripts.

Example:

- Registration (Manual Use Case)

Step1 – Insert Name
Step2 – Insert Address
Step3 – Insert State
Step4 – Click Submit button

- Login (Manual Use case):

Step1 – Insert UserID
Step2 – Insert Password
Step3 – Click Submit button

- Registration (Automated):

Step1 – Call function Registration (Registration function = Step1 + Step2 + Step3 of the Manual Use case)
Step2 – Call function ClickSubmit

- Login (Automated):

Step1 – Call function Login (Login function = Step1 + Step2 of the Manual Use case)
Step2 – Call function ClickSubmit

- ClickSubmit (Automated):

Step1 – Call function ClickSubmit (ClickSubmit function = Step3 & Step4 of the Manual Use case)

- Automated test script (Purpose – Register multiple users then login.)

GetValue Name
GetValue Address
GetValue State
Call function Registration(Name, Address, State)

Call function ClickSubmit

GetValue UserID
GetValue Password
Call function Login(UserID, Password)

Call function ClickSubmit

Hint: The words in the ( ) are the variables for the specific functions. You must pass their value to the function in order for the function to execute properly.

Steps 4 & 5 – These steps are combined because the creation of the data source and the insertion of verification points can and should be completed in tandem.

The data source (i.e., just about any coma-separated file (.csv)) stores the 'GetValue' data that is used in the automated test script. The automated test script becomes the baseline for the data

source. First, try to visualize the data source as a spreadsheet. Using the automated test script example in Step 3, the column names of the data source would be; ***Name***, ***Address***, ***State***, ***UserID***, and ***Password***. Underneath the column names is the specific data that is played back through the automated script. By adding a simple FOR or WHILE loop in the automated test script, you could rerun the same automated test script with different data up to 2 million or more times. How's that for efficiency?

In manual testing, testers are visually inspecting the screens as they go step by step through the test script. The visual inspection of the webpages is very seldom documented within the manual test script. It's assumed that the tester will use his or her experience (or common sense) to do the visual inspection. This same assumption can not be made with automated test scripts. Automated test scripts will only do what you tell them to do; nothing more – nothing less. Without verification points, the automated tests scripts can only test simple functionality.

Hint: The ability of the automated scripts to get from A to Z can prove some functionality is performing correctly. Using verification points that recognize objects rather than using verification points that take screen shots, will drastically reduce your test script maintenance.

Verification points are the 'eyes' of the automated test script. The verification points can and should be inserted throughout the automated test scripts to validate expected results. However, too many verification points will slow script playback and add to script maintenance. The way to solve this dilemma is to only use verification points when validating expected results.

Prior to the Implementation phase of PADDIS, thorough code, automated test script, and data source reviews are held. The automated test scripts are tested (and tested again) to ensure they are executing and creating the expected results correctly. Once the automated test scripts receive approval, they are placed under configuration management control to await product test.

Implementation and Support

The amount of time and effort to implement and support the automation effort will greatly depend on the success, or the lack of success, of the four previous phases. A firm understanding of your QA team's strengths and weaknesses will also play a role in the last two phases.

Implementation

'Where's the button that says GO?" asks the tester. Adequate training of the QA resources (i.e. the actual user of the automated test scripts) is mandatory. This is especially true if the automation project was outsourced. The every-day user of the automated test scripts must be able to understand, even if its on a high-level, the correlation between the manual test scripts and the automated test scripts. They should be able to add and remove data from the data source, insert verification points, analyze results, and have the ability to create the automated test scripts by themselves. Once the QA tester has gained the ability and confidence to execute, verify, and manipulate the automated testing assets, it is time to implement the automated scripts into the overall testing plan/methodology.

Support

Within the Support phase the following saying almost always holds true; "The more work and effort you put in up front, the less work and effort will be needed at the end." The Support phase of test automation should consist of the following tasks:

- Updating functions to reflect new or changed functional requirements
- Modifying verification points as the GUI and/or functional requirement change
- Adding enhancements to functions and test scripts
- Adding additional test data into the data source

Conclusion

The PADDIS methodology is not just theory. It's based on actual, hands-on experiences in automating website testing at UPS in which both success and failure was experienced. Through it all, the PADDIS methodology has been refined, enhanced, and simplified. The current PADDIS methodology has been tried and proven true through three different web application test automation efforts, automating the verification of over 75% of all functional requirements. If the PADDIS methodology is used for test automation, rather than an ad hoc, chaotic process, your QA department will be able to truly ***"do more with less"***.

References

James A. O'Brien (2000). Introduction to Information Systems: Essentials for the Internetworked Enterprise. College of Business Administration, Northern Arizona University: Irwin McGraw-Hill Publishing.

Jim Keogh (2000). Project Planning and Implementation. Needham Heights, MA: Pearson Custom Publishing.