QuickTest® Project Management: A Best Practices Guide

An introduction to organizing and managing test assets used for HP's QuickTest® Professional; by Boyd Patterson, President of Patterson Consulting, LLC.

Table of Contents

Introduction 3
What are Test Assets? 3
Types of Projects 4
Simple Projects 4
Complex Projects 4
"In-Between" Projects 5
Organizing Your Test Assets 5
Core Hierarchy 5
Module-Based Hierarchy 6
Methods of Project Management 7
Using Quality Center / Application Lifecycle Management 7
Implementing the Core Hierarchy 7
Implementing the Module-Based Hierarchy 8
Using Template Projects 8
Using Windows® File System 9
Establish Shared Network Storage 9
Utilize a Root Folder for All QTP Assets 9
Create Sub-Folders for each Testing Effort 9
Using Test Design Studio 12
Interacting with the File System 12
Implementing Your Testing Effort 12
Adding a Reference to the "Common" Testing Effort 13

Integration with Quality Center / Application Lifecycle Management 13
Using Templates to Generate Your Test Asset Hierarchy 13
The “Build Process” Keeps Your Project in Sync 14
Try Test Design Studio 15
Summary 16
About the Author 16
Acknowledgements 16
Glossary 17

Introduction

Many test automation projects fail. **Too many!**

There are many resources and discussions on the various reasons why so many projects fail, and one of the most prevailing arguments is that **test automation takes too much time**. Fortunately, the time needed to successfully automate and maintain a project can be reduced.

One of the biggest keys to a successful implementation of test automation is Project Management. Project management, as discussed in this text, refers to how you organize all the tests and supporting files that are necessary to implement your testing solution using HP's QuickTest® Professional (QTP) product.

This document describes the basics of project management, introduces proven tools and methods for managing your automation projects, and highlights the significant benefits that an organized process can yield.

Note that a glossary is available at the end of this document for many of the terms and acronyms used within the document.

What are Test Assets?

Test assets are the collection of files necessary to execute your automated tests. The primary test asset is the QTP test itself, but a QTP test can (and typically does) utilize many other files to accomplish its testing goal. Some examples of test assets include:

- A **QuickTest® Professional Test** is the primary test asset that is used to drive the testing effort. Each test is actually a collection of supporting files that help accomplish the goal of the test.
- An **Object Repository** is used to map the Graphical User Interface (GUI) elements of your application into a reference file easily accessible by QTP. Each Action in a QTP test can have its own object repository, but many projects will create and reference a *shared* object repository instead (or even use them in combination).
- A **Function Library** is a collection of reusable code that is separated from the test itself and stored in an individual file. QTP tests can then leverage this shared code without the need to repeat the logic in each test. *Effective use of function libraries is one of the biggest factors to reduce the maintenance cost of your testing effort!* A function library should be saved with the `.qfl` file extension, but older implementations may use the `.vbs` or `.txt` extension.
- A **Data Table** can be used to data-drive your test. Each QTP test automatically includes a Microsoft® Excel®-based data table as part of the test, but you may also import other data files for your test.
- An **Environment File** is an XML-based file that is used to store environment variables. This can be used to store configuration variables external to your test so that you can easily modify the environment a test will execute within without the need to modify all of your tests. Technically, any string-based data can be stored in an environment file.

- A **Recovery Scenario** is a file that defines how QTP can recover from different unexpected scenarios during the execution of a test.

The above list is an introduction to the most common QTP assets. Not all tests will require the use of all of these test assets, but most tests will use at least one. Other assets might include reference files for "before and after" comparisons, sample screen shots of the application, or any other file that is beneficial for the successful execution of your tests. All automation efforts can differ in their implementation, but it is important to ensure that **all assets needed for your test execution are organized with the rest of your testing assets**.

Types of Projects

Project implementations can vary greatly between organizations and the application under test (AUT). Organizations with simple applications or less mature automation experience tend to have basic projects, while those with complex applications or more mature automation experience will often produce complex projects.

Simple Projects

A simple project typically consists of a **relatively small number of tests** (perhaps 20 or less). These tests may or may not take advantage of shared assets like shared object repositories and function libraries.

Most new testers or test organizations tend to create simple projects. This is largely due the **lack of experience** with test automation and the maintenance that can befall a project as it grows over time. When significant changes come to the AUT, these types of projects are often "thrown away" due to the high maintenance of upgrading the tests compared to starting over.

Due to the small nature of these types of projects, project management is not a big factor in the success of the project. Even so, applying some of the concepts discussed in this text can enhance your project.

Complex Projects

Complex projects typically consist of 100 or more tests and make heavy use of shared assets. These projects can contain multiple shared object repositories and function libraries to help reduce maintenance costs.

These are the types of projects that are typically **created by mature testers and test organizations**. Careful planning must be done to ensure that common code is made reusable by all testing assets and that the tests are protected (as much as possible) by changes in the AUT. **These projects can live for several years under routine maintenance.**

Due to the large number of assets in complex projects, **project management is essential to the success** of the projects. Assets must be easy to locate, share, and maintain. Many developers will not spend more than a few minutes looking for an existing solution before creating one of their own, so making the information easily available can reduce code duplication that is common in large projects.

"In-Between" Projects

The "in-between" projects take aspects from both the simple and complex projects. This typically means that the number of tests is relatively small, but the project has implemented the use of shared assets to reduce maintenance.

These projects are typically created by testers who have been "burnt" by the failure of a simple project in the past and are seeking to improve the longevity of their work. These projects many also be created by mature organizations when the AUT or testing effort is not very complex. **This is likely the most common type of project.**

Organizing Your Test Assets

No matter which method of project management you choose, many concepts remain the same. Your **test assets must be well-organized to facilitate success**, and the same type of organization can be adapted to the various methods described later. I will begin by introducing the core organizational concepts, and then touch on how those concepts apply to the different project management methods. Later, I will discuss how you can use QC/ALM or the File System to implement this organization, as well as introduce a tool called Test Design Studio that can enhance your efforts.

Core Hierarchy

I recommend a basic hierarchical organization of assets where each type of asset (e.g. test, object repository, function library) is represented in its own folder (See Figure 1). Note that most methods use the term "folder", but can be substituted by the "folder-like" grouping mechanism of the project management method of your choice.

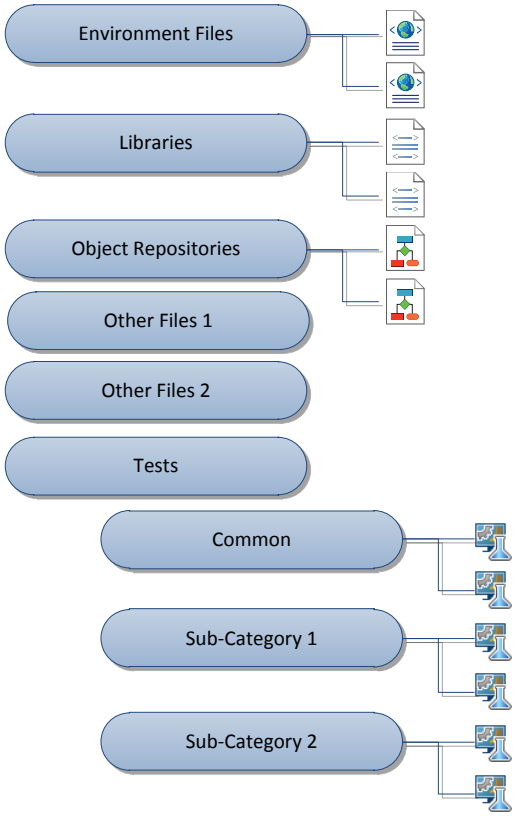

Figure 1: Core Hierarchy

The reason I recommend this approach is that users are typically searching for a particular type of asset. Having the asset type as the first level of organization allows the user to **quickly drill into the assets that are relevant**.

Tests will typically require one or more levels of organization below the base "Tests" folder as the number of tests in the project grow. This structure is up to the end-user to decide. Some common methods of organization include:

- **Test Type** (e.g. build verification, regression) – Some tests are specifically designed for a certain aspect of testing, so you can organize them as such. If you do you use a test execution tool (like QC/ALM) to manage your test sets, this type

of organization can help you keep track of which tests need to be executed for different types of testing to be performed. Note that this type of organization does not lend itself well to when a single test can participate in multiple test types since the test can only be stored under one of the test type folders.

- **Use Case** or **Requirements** – You may want to organize your tests based on the use cases or requirements you are testing.
- **Feature Area** – Most applications of even the smallest size are broken down into feature areas. In the popular "Flight Tours" sample application that is shipped with QTP, the feature areas might include "Book a Flight", "Open Order", or "Fax Order". With this type of organization, it is also beneficial to have a "Common", "Multi Feature", or "End-to-End" folder to store tests that cross multiple feature areas.

Sub-folders can be great organizational tools, but be cautious with their use in your organization effort. **A deep hierarchy of folders can make it harder to find what you are looking for.** If a folder only contains a small number of tests (i.e. 20 or fewer), do you really need a folder? Naming conventions for tests can also reduce the need for folder-based classification since the name of the test can convey the same information that might have been gained from a sub-folder.

Module-Based Hierarchy

Many enterprise-scale applications can be composed of multiple modules. A module can be viewed as a relatively large group of related features in the AUT. Most modules in an application are **primarily independent features that complement the other modules**. Example modules for an application might be: "Reporting", "Order Entry", or "Order Fulfillment". If you are familiar with QC/ALM, the modules in that AUT could be "Requirements", "Dashboard", "Test Plan", etc. Each module relies on the others, but is primarily a collection of features specific to that module.

When you are faced with a module-based application that is large enough to justify the additional organization, I recommend creating a hierarchy level above the core hierarchy for each module as shown in Figure 2.

This allows you to separate your assets by module and still maintain the basic hierarchy within each module.

You will notice a "_Common" folder listed in the Figure 2. This is a special folder that is meant to represent assets that are **common to more than one module**, and will have the

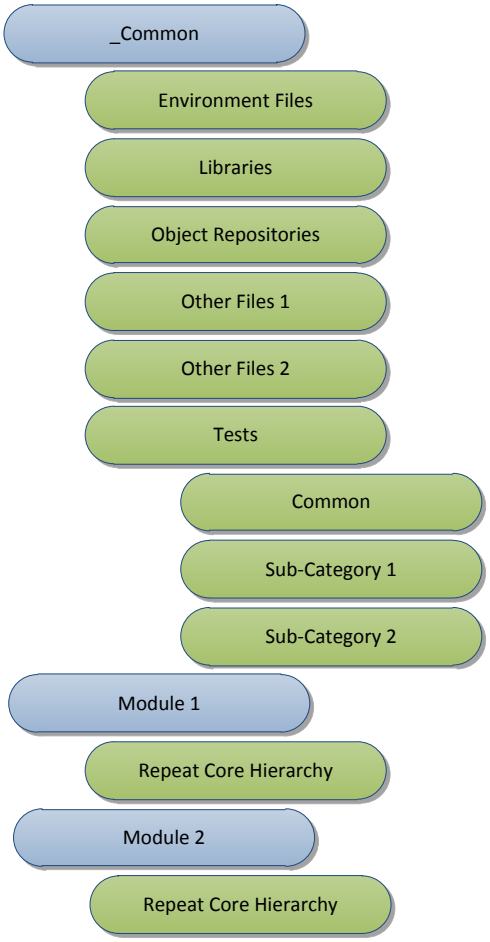

Figure 2: Module-Based Hierarchy

same core hierarchy as the other modules. Note that the reason for placing the "_" character at the beginning of the name is so the folder will typically sort to the top of the list as is the case with most management software. If your management technique allows you to specifically order the folders, the leading "_" character is not necessary.

Methods of Project Management

There are several options available to QTP users to satisfy their project management needs. Many turn to HP's complimentary Quality Center or Application Lifecycle Management (QC/ALM) products. Others on a tighter budget will use the standard Microsoft Windows® file system to organize their assets. More and more users are also turning to Test Design Studio to manage their projects whether they use QC/ALM or the file system. The following will discuss the different methods of project management and how you can apply the Core and Module-Based Hierarchies in those systems.

Using Quality Center / Application Lifecycle Management

Note that Application Lifecycle Management is the newer name given to v11 of the Quality Center product line, and the two product names are typically interchangeable in this text. Even Quality Center was previously named TestDirector®, but that version of the product is no longer supported by HP.

Over time, this product has grown from a basic test management tool into a full-featured lifecycle management tool (hence the new name), but one of the key benefits of this system remains; project management of your testing assets.

I recommend at least using Quality Center v10 or higher for project management, and this is due to the new "Resources" module that was added in v10 (renamed as "Test Resources" in v11). Prior to this new module, your testing assets had to be stored as attachments to folders and/or tests within the "Test Plan" module of QC. This method of file storage was cumbersome and not easily discovered by end-users. The new "Resources" module makes it easy to organize your different test assets, and, since these items now have a specific module in QC/ALM, the files are easily discovered. Even without the "Resources" module, you can continue to use and be successful with the prior versions of QC that are still supported by HP.

Implementing the Core Hierarchy

To apply the proposed Core Hierarchy from above in QC/ALM, all items from the "Tests" folder in the Core Hierarchy will be placed under the "Subject" folder of the "Test Plan" module. The placement of the other items will depend on whether or not you have a version of QC/ALM that supports the "Test Resources" module (referred to just as "Resources" prior to ALM).

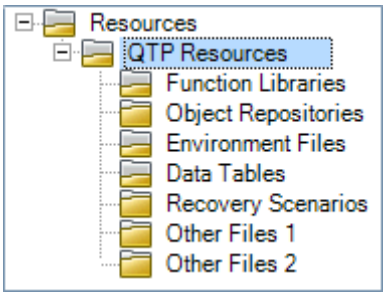

Figure 3: Test Resources Folder Structure

Using the "Test Resources" Module for Non-Test Assets

Beginning with Quality Center v10 and continuing in Application Lifecycle Management v11, there is a "Resources" (QC) or "Test

Resources" (ALM) module that is specifically designed to store all the supporting files used by your tests. From the Core Hierarchy, this includes Environment Files, Object Repositories, Function Libraries, and any other supporting files.

Since QC/ALM supports many types of automated tests, it is best to **separate your QTP-specific resources from those used with Business Process Testing (BPT) or manual testing**. Under the root "Resources" folder, create a folder called "QTP Resources" if it does not already exist. Under the "QTP Resources" folder, create a sub-folder for each file type (i.e. "Environment Files", "Object Repositories", etc.) that you plan to store (as illustrated in Figure 3).

Using the Test Plan Module for Non-Test Assets

If you are running a version of Quality Center that does not support the "Test Resources" module (i.e. Quality Center v9.2 and below), you can **use the "Test Plan" module to store all the supporting files** used by your tests. To help distinguish your resources from the other folders that store tests in "Test Plan", create a root folder under "Subject" called "Resources". This will be the primary folder where all resource files are stored. Once created, you will duplicate the folder structure described in the *Using 'Test Resources' Module for Non-Test Assets* section as illustrated in Figure 4. **Each test asset will then be stored as an attachment to the corresponding folder.**

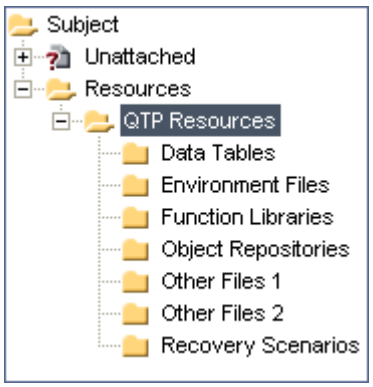

Figure 4: Test Plan Folder Structure

Implementing the Module-Based Hierarchy

If your needs require the Module-Based Hierarchy, your organization in QC/ALM will be very similar to that described under *Implementing Core Hierarchy*. The exception is that you will add one more level of organization between the "QTP Resources" folder and the individual file type folders. Create **one folder called "_Common"** to store assets common to all modules, and then an **additional folder for each individual module**. Figure 5 illustrates the "Test Resources" module setup with a Module-Based Hierarchy, and the configuration will be very similar to those using the "Test Plan" module instead. Note that while the "Module 2" folder is not expanded in Figure 5, it will display the same sub-folder structure as the "_Common" and "Module 1" folders.

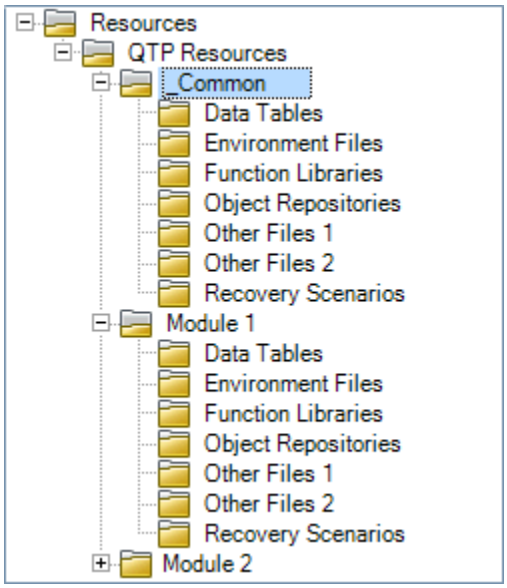

Figure 5: Module-Based Hierarchy in Test Resources

Using Template Projects

To help ensure that all new projects are setup with the proper folder structure, you can choose to implement this base folder structure within a template project, and then use that template when creating new projects. Refer to QC/ALM documentation for how to setup and use templates.

Using Windows® File System

If you do not have access to any project-management software, your only option is to organize your files on the file system.

Establish Shared Network Storage

If you work in a team-based environment or use a lab to execute your tests, you should **designate a shared network drive** for the storage of your test assets. Windows®-based security settings can be used to control access to the files.

Utilize a Root Folder for All QTP Assets

Your goal should be to store as many of your QTP assets under a single folder as possible. This makes it easy for anyone who seeks QTP-related files to begin their search. If storage space or other issues prevent you from having a single root folder, try to limit the number of folders as much as possible so that assets can be quickly and easily discovered. The remainder of this text will focus on a single root folder, but the concepts can be easily adapted to a multi-root environment. In general, you will have to duplicate all steps for each root folder. An example name for the root folder (and the name that will be referenced in this text) is "QuickTest Projects".

When working with a network storage location, you typically want to **use UNC-based file paths** for your files (e.g. `\\Server\Share\QuickTest Projects`). This ensures that all users can access the location. While appropriate, this long file path can become cumbersome, so one recommendation is to map the location of your networked root folder to a drive letter. **The key to success is that you must standardize on a single drive letter that everyone must use!** I suggest coming up with a drive letter that "means" something like "T:" for "**T**ests" or "Q:" for "**Q**uickTest". This mnemonic will make it easier for teams to remember which drive letter is used for storing QTP test assets. While using a standard drive letter can be convenient, you should still **avoid hard-coding this or any path into your test code!** This will make it harder for you to relocate your root folder in the future. Server names can change, and drive letters may be repurposed beyond your control. In other words, you can almost expect that your root folder will be moved in the future. In your code, you should **define a constant value that represents your root folder**, and then always reference that constant for your path. If the path ever changes, you will have a single point of maintenance by only needing to update the definition of your constant.

Create Sub-Folders for each Testing Effort

Under the root folder, you will need to **create a sub-folder for each testing effort**. How you measure the "testing effort" is essential to this level of organization. Typically, each AUT would qualify as an individual testing effort, but you may find that a single testing effort covers multiple AUT's. You may also define some testing efforts that are not related to any AUT (as discussed by the "Common" and "Template" testing efforts discussed later). Figure 6 illustrates the folder structure of a testing effort called "Flight Tours".

Defining the "Common" Testing Effort

In the typical life of an automated tester, he/she will find that some work effort is so global in nature that you find yourself doing the same type of work in multiple projects. This is exactly the type of work that comprises the "Common" testing effort. All organizations should start early to **build a library of commonly-used functionality**. This includes defining one or more function libraries that make it easier to perform work in other projects. Examples of the type of functionality in these libraries might be value conversion functions, string manipulation functions, or custom reporting functions. This is functionality that might be useful to any testing effort, and so it must be **stored outside of any specific project**.

Create a sub-folder under the Root Folder called "Common" (or whichever term you prefer). Under the "Common" folder, you will then want to create the sub-folders for the Core Hierarchy.

I recommend that the first file in this testing effort be a file called "Common.qfl" stored under the "Libraries" Core Hierarchy folder. This will be a function library that can be used for defining common constants and functions, and the first entry can be the declaration for the root projects folder as discussed in the section *Utilize Root Folder for All QTP Assets*. For example:

```
Public Const QTP_PROJECTS_FOLDER = "\\Server\Share\QTP Projects\"
```

You can continue to build out this library as additional items require declaration.

Defining Additional Testing Efforts

Any individually recognized testing effort should be organized under its own sub-folder. The name of the sub-folder should clearly identify the effort. Often, the testing effort is defined by the AUT (e.g. "Flight Tours" for the sample application delivered with QTP). You might find that within your organization, you have testing efforts that span multiple AUT's.

When you have a simple testing effort, all of the Core Hierarchy folders are created as sub-folders of the folder you created for your testing effort (see Figure 6).

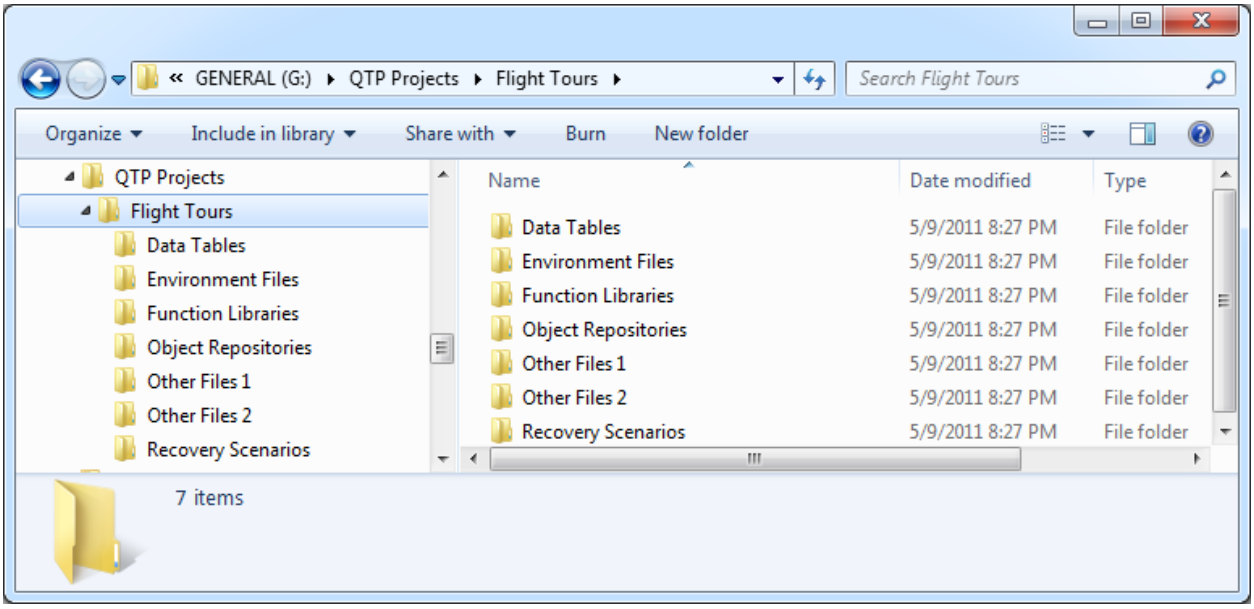

Figure 6: Core Hierarchy on File System

When you have a testing effort that **spans multiple AUT's or perhaps multiple modules**, you should consider using the Module-Based Hierarchy for defining your sub-folders. Figure 7 illustrates the Flight Tours application where the testing effort is separated into two modules; Desktop Application and Web Application.

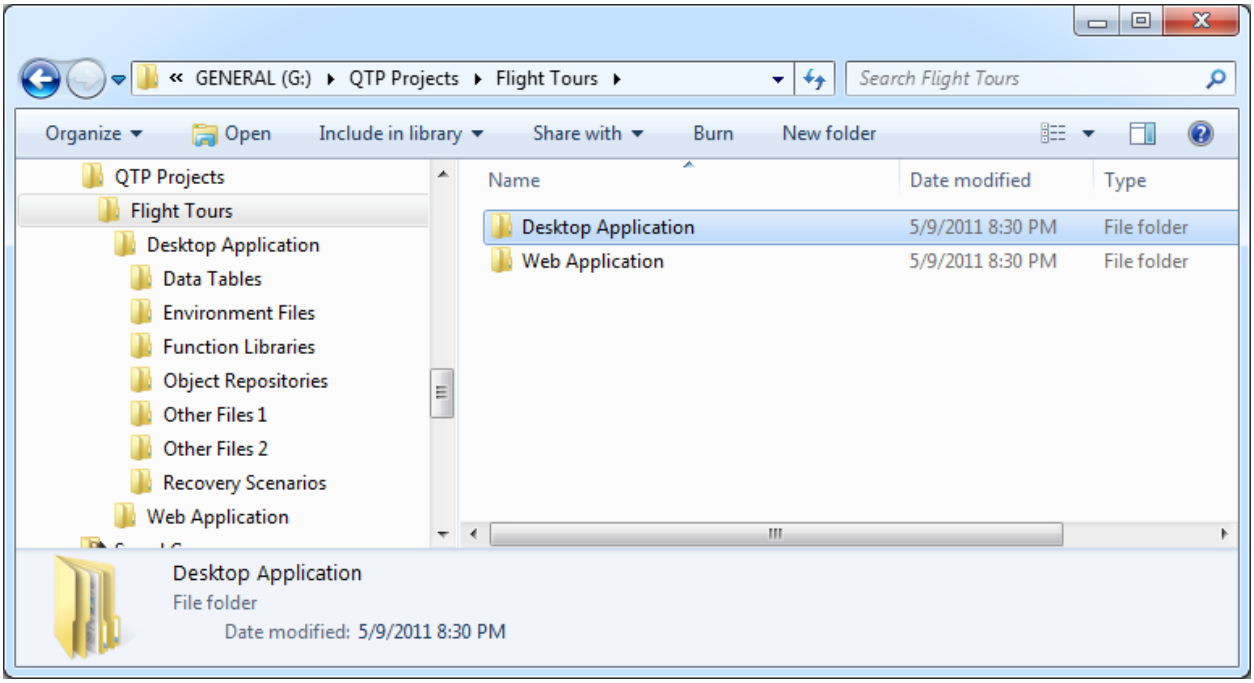

Figure 7: Module-Based Hierarchy on File System

The same folder structure shown under "Desktop Application" is repeated under "Web Application".

Creating a Template for Testing Efforts

To quickly create a new testing effort, you can **build a template** of the folder structure and any files that are typically used. When you need to create a new testing effort, you simply copy and paste the template while giving the pasted contents a new name. You can create a single template used by all new testing efforts, or create specific templates for different types of testing efforts. Either way, basing your new folder and file structure off of a template will help your implementation **remain consistent** and **speed the process of creating your supporting structure**. I recommend you create a "Templates" testing effort that houses all of your available templates.

Using Test Design Studio

Test Design Studio is a tool by Patterson Consulting, LLC that offers many features to **improve the life of testers** working with QuickTest® Professional. One of those many features is project management. The following will discuss how you can improve upon the base project-management features of the Windows® file system and QC/ALM.

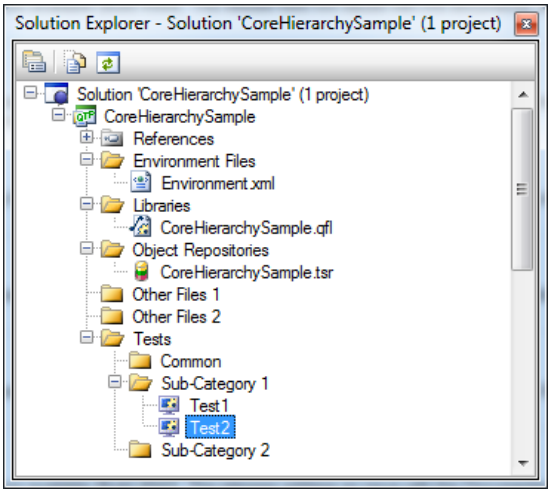

Figure 8: Test Design Studio Solution Explorer

Interacting with the File System

As outlined in the section *Using Windows® File System*, you can easily take advantage of native file system support to help organize and manage your project. Test Design Studio was designed to **directly interact with the file system** while still providing a **managed view** of your project.

The heart of project management in Test Design Studio is the 'Solution Explorer' tool window shown in Figure 8. As you can see, there are folders for all the groupings of the Core Hierarchy. The folders and files you see listed in the 'Solution Explorer' are **direct representations of the same folders and files on the file system**. Deleting, copying, and moving items in 'Solution Explorer' will also modify the underlying file system accordingly.

One of the benefits of the 'Solution Explorer' tool window is that you have a **single, easily navigated view into all the testing assets** that are part of your testing effort.

Implementing Your Testing Effort

In Test Design Studio, each testing effort is managed by a single solution. In Figure 8, the solution is called 'CoreHierarchySample' and is displayed at the top of the hierarchy. Within each solution, you create one or more projects that will be based on the Core Hierarchy. If you are basing your testing effort on the Module-Based Hierarchy, you will create a project for each module. Otherwise, you will create a single project (often given the same name as the solution) that will manage the Core Hierarchy (Figure 8 shows a solution based on the Core Hierarchy). This allows you to **keep you test assets organized while still benefiting from the single view** provided by 'Solution Explorer'.

Adding a Reference to the "Common" Testing Effort

As previously mentioned, I recommend creating a "Common" testing effort that contains very generic functionality that could be applied to all of your targeted testing efforts. Just as you did with your standard testing efforts, you will **create a solution and a project for your common testing effort**. The solution and project will follow the Core Hierarchy, so that means there will be a solution with a single project.

Once the project is created, you can then **add the same project to the solution used for your targeted testing efforts**. This will allow you to see the "Common" test assets along with your AUT-specific test assets. Everything will be visible in the 'Solution Explorer' tool window. At this point, you can begin to see the benefits that Test Design Studio offers over using just the file system because the 'Solution Explorer' **only displays the test assets that are relevant to your testing effort** and can bring together data that is scattered in different locations. You do not have to filter through the other folders on the file system that contain test assets for other projects. Everything you need is displayed in a simple, easy-to-browse tool window!

Integration with Quality Center / Application Lifecycle Management

Test Design Studio is designed to work in conjunction with the file system, but it also has support for files stored in QC/ALM. When you add an existing file to your project, you can choose between files on the file system or those stored in QC/ALM. When you choose a file from QC/ALM, **a local copy of that file will be downloaded to the file system and managed by Test Design Studio**. You will be notified if the file on the server has been modified so that you can keep the local copy up-to-date. Having a local copy also allows you to **work offline from QC/ALM**, which can extend your productivity when a connection to the server is not available.

Using Templates to Generate Your Test Asset Hierarchy

One of the recommendations for working with the file system was to create a series of templates that could easily be copied and pasted to quickly setup a new testing effort. Test Design Studio **offers a template feature that is much more powerful**.

Within Test Design Studio, you can **setup very complex templates** to represents solutions and projects. These templates allow you to specify the folders and files that will be included when the template is generated. Instead of a "copy and paste" procedure, **each template is processed at the time of creation to dynamically generate content**. For instance, you can use the name of the project to determine the name you give to the function library. For the "Flight Tours" sample discussed earlier, this would allow you to create a project called "FlightTours" and generate corresponding test assets like the "FlightTours.qfl" function library and the "FlightTours.tsr" object repository. Dynamic processing does not stop there. Each file can also be processed for dynamic creation! This allows you to **generate context-sensitive data directly within the files**.

Test Design Studio comes pre-packaged with a QTP project template that models the projects used by the Core Hierarchy and the Module-Based Hierarchy. You can customize these templates to meet your specific needs, or you can create new templates instead.

The "Build Process" Keeps Your Project in Sync

One powerful and time-saving feature of Test Design Studio is the "Build Process". When applied to projects based on QTP, this process **performs some key operations to keep your projects in sync**. For each test that is part of a project, the build process will ensure that the test has references to all the libraries and object repositories that are defined in your solution and project!

When you have a lot of tests, adding a single library to your test assets can be a cumbersome process. Not with Test Design Studio! Once you add a new library, you simple initiate the "Build Process". Every test in your project will be updated to include the new library. This process works with tests stored on your file system or in QC/ALM, so everyone can benefit!

The "Build Process" enables the freedom to **create a well-organized project without the burden of keeping your QTP test references updated**.

Consider the following scenario to see how much time the Test Design Studio "Build Process" can save you. If you have to update a QTP test with a new function library, you must perform the following steps:

1) Spend time browsing for the Test you want to update and open in QTP
2) Select 'File -> Settings' from the main menu
3) Select the 'Resources' tab
4) Click the 'Add Resource' button
5) Click the 'Browse' button
6) Spend more time browsing for the function library you want to add, and then selecting it.
7) Click 'OK' to confirm your changes.

These seven steps do not sound like much work, but quickly become tedious and time-consuming as you have to update all the tests in your testing effort. All the menu options and file browsing can easily take several minutes per test, and this time adds up quickly as you have 10, 50, 100, or even more tests in your testing effort! Perhaps most importantly, **this is time your testing resource is wasting on non-value-add activities**. Many of us became automated testers to avoid redundant manual tasks, and this task is about as mind-numbing as they come.

The Test Design Studio "build process" is **fully automated and takes a fraction of the time**. Since Test Design Studio uses the QTP Automation Object Model, it performs the same tasks you would execute manually. Each test is updated quickly since there is no user-interface to manipulate. Perhaps the best aspect is that your testing resource is not tied up during the update, so that resource can focus on tasks that **add real value to your automation effort**!

The chart in Figure 9 helps illustrate the effort it takes to update tests with just one function library using either QTP or Test Design Studio. As you can see, the time-savings quickly become significant as you have more and more tests in your testing effort.

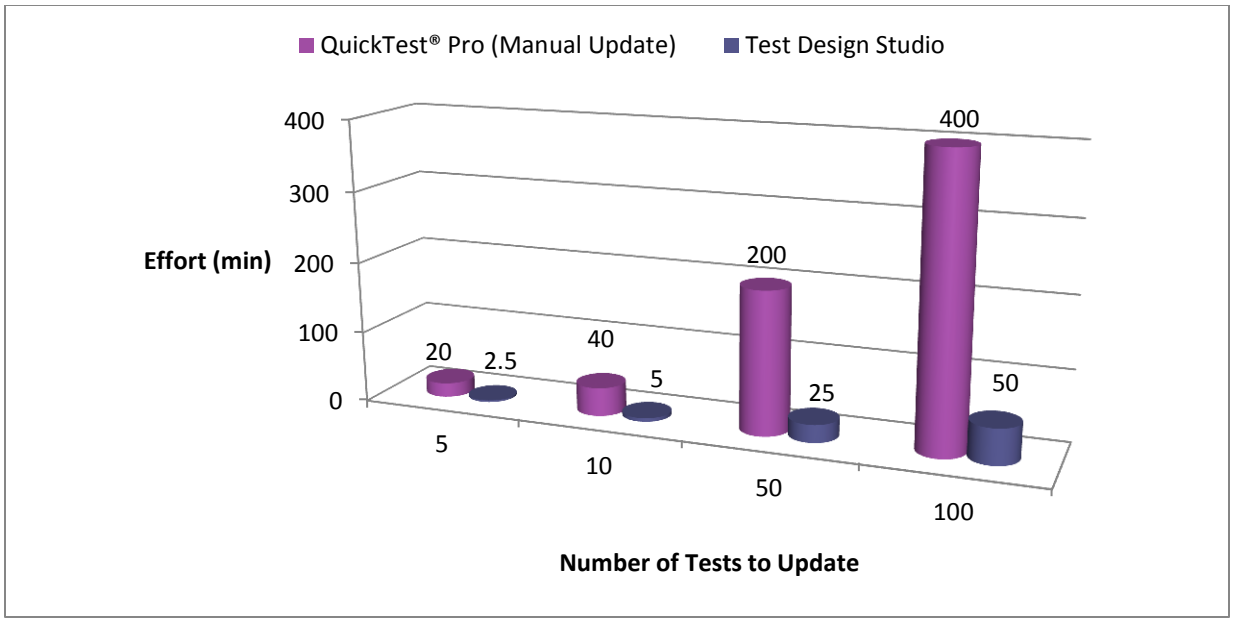

Figure 9: Effort to Update Tests

Just look at the amount of time you can save using the Test Design Studio "Build Process" and this time savings comes back in every instance where you need to add/remove function libraries or object repositories from your testing effort.

Try Test Design Studio

While Test Design Studio excels at project management, that is just one of the many great features of by this tool. Test Design Studio is available to download and includes a 20-day, fully functional evaluation period. You can find more information about the tool, ROI calculators, feature videos, downloads and more from the following website:

http://www.patterson-consulting.net/tds

Summary

Managing your QTP projects does not have to be difficult. Using QC/ALM, the File System, or Test Design Studio, you can organize and your project in a way that is efficient and easy to comprehend. Tools like Test Design Studio can help you get more out of the standard implementations available by just using QC/ALM or the File System alone.

Once you apply a project-based methodology to your testing efforts, you will **open the door for new successes** in automation and **improved sustainability** of your work.

I hope you have enjoyed this discussion on project management and can implement some of the techniques discussed. Comments are welcome and encouraged! If you would like to share your own processes or discuss these techniques further, I would love to hear from you. Please feel free to e-mail me at bpatterson@patterson-consulting.net.

About the Author

Boyd Patterson is the President of Patterson Consulting, LLC. He has been working with test automation since 1999, and began his career using Mercury WinRunner® 6.0 and TestDirector® 6.0. He has continued to use the Mercury/HP suite of tools throughout his career, and eventually transitioned from WinRunner® to QuickTest® Professional 6.5. During his time in test automation, he have seen TestDirector® transition from a client-server solution to the current web-based offering of ALM. He has also seen QTP grow as a more and more powerful tool as key features from WinRunner® were transitioned to QTP with each release, and recent releases have finally been able to focus on driving new functionality.

Aside from his 12 years in test automation, he has spent the last 8 years studying software development, architecture, and design principles. During that time, it became clear to him that test automation was its own special software development activity, but the tools and the industry failed to treat it as such.

He set out to combine the benefits of software development methodologies with what he was doing in QTP, and one of the results was Test Design Studio. This product is the culmination of many years of experience in both fields, and brings the benefits of software development to test automation.

Acknowledgements

I would like to thank Paul Grossman for reviewing this document and offering feedback.

Glossary

The following glossary discusses terms and acronyms that were used by this text.

Action: When in reference to QTP, this is an independent component of a test. All tests have at least one action, but additional actions may also be created that are "called" by the main action. Actions are similar to function libraries.

ALM: See *Application Lifecycle Management*

Application Lifecycle Management: HP's Application Lifecycle Management product. This product was previously called Quality Center, but was renamed with v11 of the product line.

Application Under Test: This is the application that is being tested and may collectively refer to one-or-more applications that are part of a testing effort.

Asset: Any file or resource that is necessary for the successful execution of a test.

AUT: See *Application Under Test*

BPT: *See Business Processing Testing*

Business Process Testing: An add-on feature of QC/ALM that provides a way to organize and execute tests based on a business process.

Data Table: A collection of data (typically in Microsoft® Excel® format) that can be used to control execution of or data drive your test. Files typically have the `.xls` file extension.

GUI: The graphical user interface of your test.

HP: Refers to the company Hewlett-Packard that is the vendor for QuickTest® and the Quality Center product-line.

Object Repository: A file that is used by QTP to map your GUI screen elements into objects that are easier to interact with through the automation tool.

Environment File: An XML-based file that is used to store textual environment variables that can easily be changed to reflect the current testing environment.

Feature Area: A basic feature of an AUT. *See Also Module.*

Flight Tours: This is a sample application distributed with QTP to help drive many of the samples that are provided with QTP. It is a simple application designed to book airline flights.

Function Library: A single file that contains a collection of code elements that can be directly referenced and used from other files. These files typically have the `.qfl` file extension, but may also have the `.vbs` or `.txt` file extension.

Mercury Interactive (aka Mercury): The original product owner of WinRunner®, QC/ALM, and QTP. Mercury was eventually acquired by HP.

Module: A group of features in an AUT that are easily grouped together and complement the features of other modules in the AUT. *See Also Feature Area*.

Project: A term used by Test Design Studio to organize individual modules in a testing effort. *See Also Test Design Studio, Solution, and Testing Effort.*

QC: See *Quality Center*

Quality Center: HP's Quality Center product. This product was previously named TestDirector®, but was renamed as Quality Center in the v8.0 refresh of the product line. This product has since been renamed again as Application Lifecycle Management (ALM).

QuickTest® Professional: HP's functional automated testing tool.

QTP: See *QuickTest® Professional*

Recovery Scenario: A file that defines how QTP can recover from different unexpected scenarios during the execution of a test. Files typically have the `.qrs` file extension.

Refactor: This is the name given to the process of renaming and restricting code elements typically with the goal of adding new functionality or improving code quality.

Solution: A term used by Test Design Studio to organize a testing effort. A solution is a collection of one or more projects. *See Also Test Design Studio, Project, and Testing Effort.*

TDS: See *Test Design Studio*

TestDirector®: The previous name of the Quality Center and Application Lifecycle Management product-line.

Test Design Studio: A product by Patterson Consulting, LLC that can be used for project management as well as offering many other features useful for automated testers. More information on this product is available at http://www.patterson-consulting.net/tds.

Testing Effort: A term used to organize the high-level effort that must be performed to accomplish a testing activity. This is typically related to an AUT, but may include other self-contained testing activities like shared function libraries.

WinRunner®: A functional testing tool originally developed by Mercury Interactive that was later acquired by HP. The tool was later discontinued by HP to focus on QTP as the single functional testing tool.