

Test Design Studio

Improve the Quality of your Code

If you write QuickTest® Professional test scripts or libraries, you need to realize that you are a programmer! Even though these scripts are used for testing purposes, rest assured that you are creating program code.

Just like the program code that goes into the applications under test, the code used for test automation needs to possess similar quality characteristics. The sooner you treat test automation as a development activity, the sooner you will realize the cost-saving potential of this technology!

WHAT IS CODE QUALITY?

Code quality, as it pertains to automated tests, is often thought to be a subjective measurement of some of the following code characteristics:

- **Ease of Maintenance** - how easily can your code be maintained when the AUT is modified or new functionality is required?
- **Low Complexity** - similar to readability, how easily can someone comprehend the purpose of your code?

BENEFITS AT A GLANCE

- **Real-time Syntax Checking** - helps you identify syntax errors quickly, as they are created
- **Static Language Analysis** - helps you identify potential code issues and run-time errors
- **Code Metrics** - helps you objectively measure the complexity and maintainability of your code

Test Design Studio

Our Test Design Studio project was the first product on the market to treat QuickTest® Professional automation as a programming activity. The rich integrated development environment allows automation engineers to reach new levels of productivity. With the introduction of code quality features in our latest version, you can now reach new levels of quality in the code you are creating.

Real-time Syntax Checking

Test Design Studio identifies errors in real time as you type! All errors are underlined in the editor so that you can quickly identify the problem and fix it. QuickTest® Professional performs syntax checking when you save a file or think to invoke it off manually, but Test Design Studio is constantly analyzing your syntax for immediate feedback.

*Test Design Studio is Constantly
Analyzing Your Syntax for
Immediate Feedback*

Which Would You Choose?

Compare the two screen shots below to see the difference between the editing surface in QuickTest® Professional compared to the feedback you receive in Test Design Studio. Given the exact same code in both tools, which do you think will make it harder to produce syntax errors?

```
1: Option Explicit
2: Dim duplicateDeclaration
3:
4: Public Function MyFunction(ByVal a, ByRef b, neverUsed)
5:
6:     If a = b Then
7:         a = b + 1
8:         Reporter.ReportEvent micFailed, "This step failed"
9:         Exit Sub
10:    End If
11:
12: End Function
13:
14: Dim duplicateDeclaration
15:
```

Screen Shot from QuickTest® Professional v10.0

```
1: Option Explicit
2: Dim duplicateDeclaration
3:
4: Public Function MyFunction(ByVal a, ByRef b, neverUsed)
5:
6:     If a = b Then
7:         a = b + 1
8:         Reporter.ReportEvent micFailed, "This step failed"
9:         Exit Sub
10:    End If
11:
12: End Function
13:
14: Dim duplicateDeclaration
15:
```

Screen Shot from Test Design Studio v2.0

Test Design Studio

Static Language Analysis

Syntax checking is only half the story, and only ensures you follow the basic syntax of the language. Other errors are typically not revealed by QuickTest® Professional until you try to execute your tests. This is because the code is syntactically correct even if not implemented properly. These are the most time-consuming errors to fix because it is only during execution that you discover many silly mistakes. Oversights like declaring the same variable more than once because you copied/pasted code, forgetting to declare variables when 'Option Explicit' is used, or mistyping the name of a function or variable. When these issues are discovered at run-time, it usually means a significant loss of productivity. Tests have to be re-executed and application state must be restored.

Test Design Studio analyzes your code for many common errors and warns you about other potential logical errors. These issues are identified during design at the time of creation, not later when the code is actually executed and potential time has passed since the developer's mind was fresh. The following lists highlight many of the errors and warnings generated by Test Design Studio by static language analysis:

General:

- Promote use of the 'Option Explicit' statement to help enforce language rules.
- Cannot make duplicate declarations of variables in the same scope.
- Must use the various Exit statements in the proper context (i.e. using 'Exit Function' only within a 'Function' declaration).
- Check for proper use of parenthesis when invoking a function.
- Function calls must provide the proper number of arguments.

- Function calls and variable usage must refer to a known entity (identifies use of invalid or misspelled items).
- Ensure that object-based assignment statements use the 'Set' keyword, and non-object-based assignments do not.
- 'Select Case' constructs must have at least one 'Case' statement.

Class Declarations:

- Ensure classes instantiated with 'New' keyword are located in the same file as the statement instantiating it.
- Default Properties/Functions must be 'Public'.
- Only one member of a class can be 'Default'.
- A default property can only be defined on the 'Get' declaration.
- 'Class_Initialize' and 'Class_Terminate' cannot have arguments.
- Property declarations must have consistent argument signatures.
- Warn if public variables are used in a class instead of public properties.

Function/Sub Declaration:

- 'Sub' declarations must not attempt to return a value.
- Warn if a 'Function' declaration has no return value (did the developer forget?).
- Warn if a parameter is declared but never actually used.

When these issues are discovered at run-time, it usually means a significant loss of productivity

Test Design Studio

Code Metrics

Code Metrics are a useful tool implemented by Test Design Studio to provide an objective analysis of the complexity of your code. Higher code complexity leads to higher defect rates and decreased maintainability. The following metric values are calculated for major language elements including entire tests, class declarations, functions, and properties

Cyclomatic Complexity

This metric measures the number of paths through your code. Inclusion of branch and loop statements (like 'If' and 'For') increases the number of paths, and more paths lead to increased complexity.

Lines of Code

This metric counts the number of executable lines of code while ignoring whitespace and comments.

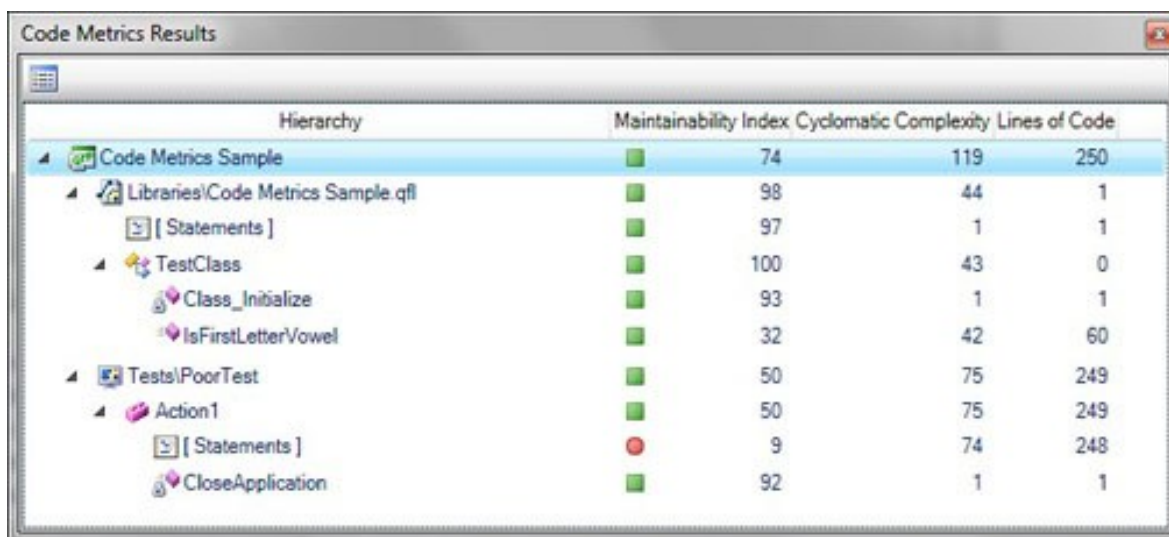
Combine Code Metrics with Peer Reviews or Outsourced Projects to Quickly Identify Complex Code

Halstead Metrics

The Halstead complexity measures are software metrics introduced by Maurice Howard in 1977, and have proven their value even as technology transforms. The individual metric values are calculated behind the scenes to help determine the consolidated Maintainable Index value.

Maintainability Index

All of the previous metrics are used to calculate a maintainability index between 0 and 100. Values of 0-9 indicate high maintenance, 10-19 moderate maintenance, and 20-100 are low maintenance.



Hierarchy	Maintainability Index	Cyclomatic Complexity	Lines of Code
Code Metrics Sample	74	119	250
Libraries/Code Metrics Sample.qfl	98	44	1
[Statements]	97	1	1
TestClass	100	43	0
Class_Initialize	93	1	1
IsFirstLetterVowel	32	42	60
Tests/PoorTest	50	75	249
Action1	50	75	249
[Statements]	9	74	248
CloseApplication	92	1	1

Sample Output of Code Metrics from Test Design Studio

Test Design Studio

Find Out More

For more information on Test Design Studio, visit http://www.patterson-consulting.net/products/test_design_studio/Default.aspx.

Download a Trial

To try Test Design Studio for yourself, download a fully-functional trial and learn how much easier automation can be when you have the right tool for the job.



PATTERSON
CONSULTING

<http://www.patterson-consulting.net>

Copyright © 2010 Patterson Consulting, LLC. All rights reserved.

QuickTest® Professional is a registered trademark of Hewlett-Packard Development Company, L.P. All other marks and names mentioned herein may be trademarks of their respective companies

Item No: TDS_20_DS_CQ_R1