TestComplete 3.0 Overview for Non-developers

Table of Contents

Introduction

I notice that on our TestComplete web pages, there is a great deal of talk about processes, automation, scalability testing, distributed testing and so forth. What is all this? What if you're not a programmer – or what if you're not even a QA professional, but you've been tasked with finding a software testing tool that meet your needs, and you've happened upon our product, TestComplete?

The purpose of this paper is to demystify the developer-speak surrounding TestComplete. After all, TestComplete was designed to be used by both developers and non-developers alike. It just happens that we tend to end up talking about the product in developers' terms. I'll now attempt to give a common sense introduction to TestComplete 3.0, keeping to a minimum of programmer-lingo.

If you're reading this online, then find the word Next up at the top of this page, and you can read this like you're reading a book. Click *Next* to turn the page.

If you've downloaded the PDF and are reading that, then just keep scrolling down...

About TestComplete

What is TestComplete?

TestComplete is an automated software testing solution. It helps you to make sure that there are fewer (or hopefully, no) bugs in your software before you release it.

Ok, so what is an "automated software testing solution"?

The vast majority of software available today (if it is tested at all) is tested by hand. Human testers try to find problems in a piece of software before it is released. However, human-only software testing is known to be slow, inefficient, tedious (and therefore occasionally unreliable.) TestComplete allows you to automate certain redundant testing tasks, thus combining computers and humans in the testing tasks. Computers (using TestComplete) can do all the boring, repetitive stuff, freeing humans up to go find new, more interesting bugs.

So "test automation" means that the computer just clicks on stuff?

No, there's quite a bit more to it than that. At least with TestComplete, there is. User interaction simulation (clicking on stuff) is just one small part of a complete test plan. This paper will describe some of the many things that TestComplete contributes to a total test plan, but for now it's enough to say that good software testing automation should test many things, including how the UI works, in a way that is easily expandable, unobtrusive and run without supervision on a daily (or nightly) basis as needed.

What is TestComplete not?

TestComplete is not a "silver bullet" which can solve all your software problems with a single button click (although I like to think it comes close). A good test plan requires a lot of work and coordination with many participants before, during and after your testing runs no matter what tool you use.

Even though TestComplete is a test automation tool, it does not automatically know how to test your software; you will have to tell it what to test and what your test criteria are.

It's also not an inflexible robot. Many think that test automation means creating very specific tests

that can never be altered without completely rewriting them. That's not true of TestComplete; it supports the ability to expand your test suites by simply adding a few lines of text to a file (that's called Data-Driven Testing (see Types of Testing (see page 4)) and it's a must-have for testing automation).

Basics

Test Automation

So let's start with some basics. As I've said, test automation is more than just automatically clicking on things, but that's as good a place to start as any.

TestComplete allows you to simulate a user who's interacting with your application. In other words, you can record interaction (see figure 1) with your tested application's user interface (click this, type that, drag something and drop it over there, etc.). Later, TestComplete will play back those recordings and make sure that your application behaves as it should.

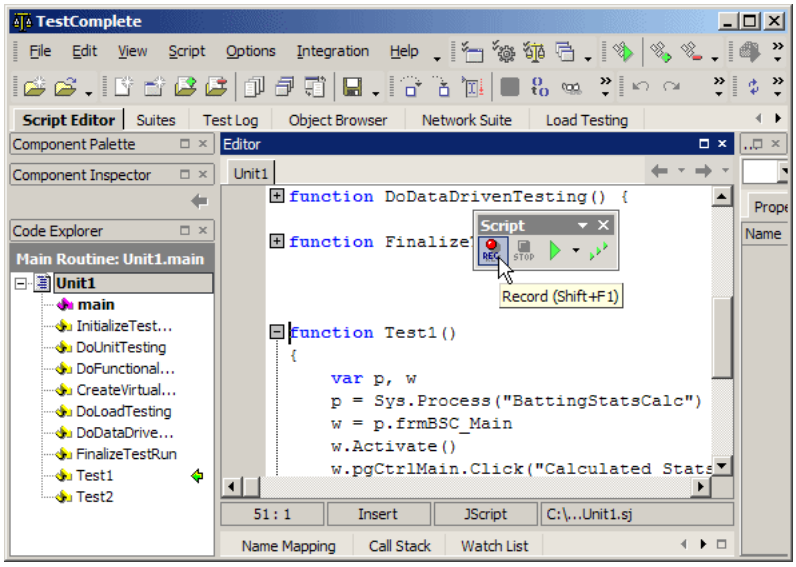

Figure 1 – *Clicking the Record button puts TestComplete into script recording mode where it will record your UI interaction. It also pops up the toolbar in Figure 1a:*

Figure 1a – *this toolbar appears while recording scripts. It lets you pause and stop recording, take screenshots, insert comments, record HTTP actions (for web server testing), etc.*

These UI actions you've recorded are managed via scripts. Scripts are something that you'll hear a lot about; they are basically little programs (see figure 2).

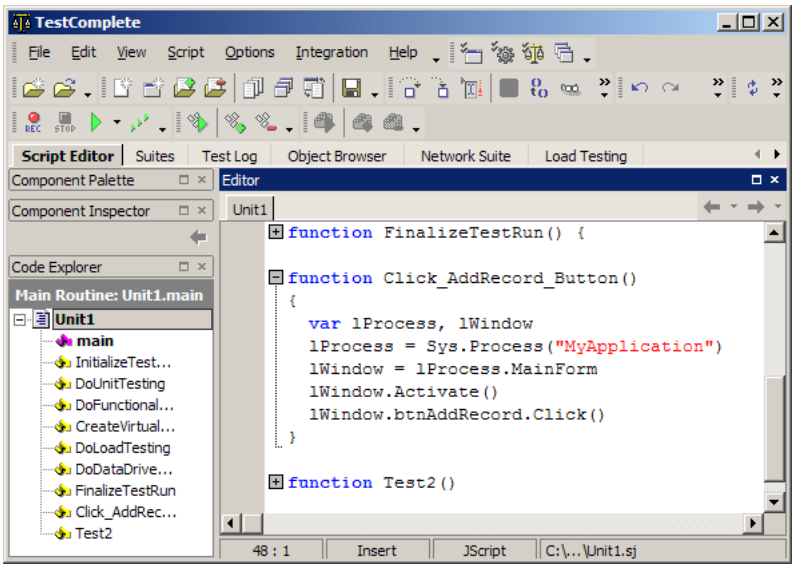

Figure 2 – *a TestComplete script (in JavaScript); you are a programmer and you didn't even know it.*

Scripts

Like it or not, ultimately automated software testing comes down to programming. There will have to be code written to test your applications. Fortunately, however, TestComplete makes it about as painless as is possible. We provide many visual and computer-aided ways to create, organize and run these scripts which will test your programs.

A large test project might be made up of hundreds or thousands of scripts, each one of those a specific test case, or a UI instruction (like Open A File), or an instruction for TestComplete to compare two screenshots for instance (see below for a discussion of comparisons). All of these scripts can either be managed in code or, if you're not a programmer, you can use TestComplete's various visual tools for managing these large projects.

Comparisons

Comparisons are at the heart of software testing. In the simplest definition, software testing is described as comparing an actual set of results to the expected results. In other words, if your application generates text file output of some sort, you would compare the actual text output of test run with a text file you know to be good – a *baseline.*

Since comparisons are such an important part of testing, TestComplete has robust support for various comparisons, and tools to make generating those comparisons easier (you knew I was going to say that, yes?) TestComplete offers the ability to compare bitmaps (or screenshots), files or application objects. Each of these different types of elements can be stored in your test project. This allows you to tell TestComplete at any time to compare a file that was just created with, for example, file number 7 that you have stored in your project.

Bitmaps – for bitmap comparisons, TestComplete compares the images pixel by pixel looking for differences. If your goal is to compare screenshots, this is made very simple by TestComplete; select a UI element in the application you're testing, and tell TestComplete to store it as a "region." TestComplete will store the bitmap and generate the code used to compare that screenshot to later screenshots of the same UI element – you don't have to write a line of code!

Objects – these days, all Windows applications are made up of objects and more objects. TestComplete can "see" those objects (see White-box Testing (see page 7)) and can therefore compare

them as easily as it compares images. Just like you did with the screenshot, you can select an object in your application and tell TestComplete to store it. It saves that object in its current state and generates the code to compare the object for you.

Files – you can tell TestComplete to store files and tell it with a single line of code to compare. If the files don't match, you can have TestComplete launch your favorite file difference utility to show you where the differences are.

Types of Testing

Functional Testing (UI Testing)

I've been stressing the point that TestComplete does far more than just traditional UI testing. But it still does do UI testing, and quite well. Traditional UI testing done by humans is basically implemented as "act like a user and see what breaks." Testers will use the software under test and make sure it behaves as it should.

TestComplete automates this process by enabling you to record UI interaction and play it back; thus simulating a user. But the process is improved by the fact that 1) TestComplete can play the same scripts over and over without getting bored (unlike a human tester) and 2) TestComplete can see into the application and make sure that the application is behaving internally as it should be, something that a human tester could never do.

In addition, TestComplete offers support for Data-Driven Testing which allows you to extend your UI Testing test cases without recording new scripts, making it a big time and expense saver.

Unit Testing

There's a lot of talk about unit testing these days. What is it? Basically, unit testing is the process of making sure the inner-most application code works as it should. This is different from UI testing in that unit testing has to be done in code. Imagine you're writing a calculator program. Part of your application's code will add numbers together. Some of your unit testing code will make sure that 2 + 2 always equals 4. It's a way of finding bugs that are buried deep in your application's code and might not be visible to a UI tester. Also, if for some reason future code changes accidentally cause 2 + 2 to equal 5, your unit testing code will quickly catch that problem.

TestComplete supports unit testing in a variety of ways, but because unit testing is typically the responsibility of programmers, most of those ways are code-based. However, TestComplete 3 now supports a visual way (see figure 3) of adding unit tests to your comprehensive testing project. If your programmers have written unit tests into their code, you can tell TestComplete to run those unit tests as part of your project. In this way, all of your testing is combined into one result log.

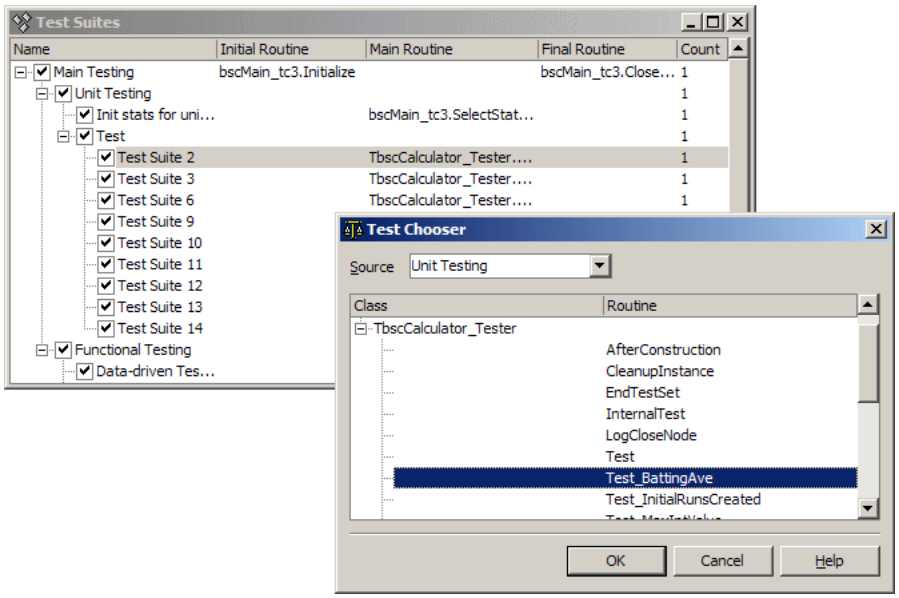

Figure 3 – *You can use the Test Chooser to select from a list of unit tests that TestComplete finds in your tested application.*

We also have some papers on unit testing with TestComplete in the technical papers section of our website.

Regression testing

Regression testing is basically the process of making sure that what has once worked has not been broken. Many times the process of fixing a bug can cause another fix to become "un-fixed". If every time an issue is identified and fixed a corresponding regression test is created for that issue, then if that issue ever resurfaces it will be spotted immediately.

HTTP Performance Testing

HTTP performance testing with TestComplete entails load testing, stress testing and scalability testing. I'll spare you the details, suffice it to say that you'll be able to create a host of "virtual users" who can carry out HTTP requests (like browsing a web site and entering information) en masse in order to pound on your web server and see the results.

Distributed Testing

This type of testing allows you to test client/server or multi-tier applications. TestComplete can run synchronized tests from multiple network clients to simulate multiple users working on a networked application.

Database Testing

Database testing is something that a lot of QA professionals need to do, but few really know how to implement it. One problem that a lot of testers run into when automating the testing of their database application is that of validating the test database by reading it through the application itself. In other words, they'll make sure that the database is as it should be by accessing the database through the data controls of the application under test. This is typically not a good idea and should be avoided. Fortunately, TestComplete allows you to directly access databases via Microsoft's ADO or Borland's

BDE technologies. You can access the data in your database without ever launching the test application.

Data-driven testing

For anyone planning on a serious testing plan, data-driven testing is a must. Here's why: when you record test scripts, they can be rather inflexible (click only this button, type only this text – just what was recorded.) That's ok if you don't need to be able to expand your tests on a whim (very few of us fit that description). A more robust way to handle testing is to record your scripts, and then alter them so that each script can do a number of things depending on what data you give them. For instance, rather than having a script that types "Joe Smith" every time it's run, you could have that script types what ever text you give it each time you run it. So one time it might type "Joe Smith" another time it might type "Jane Doe", etc. If you then have TestComplete read a series of values from a text file, a database, or a spreadsheet, you have data-driven testing. You can have TestComplete run that script over and over for each value in the text file it finds. The advantage of doing testing this way is that it is flexible and easy to extend. You can test a wide variety of test cases using only handful of scripts and when you want to extend the test cases, you only have to add a line of text or a spreadsheet entry rather than recording or writing more script code.

Visual Test Development

I've mentioned that TestComplete is for developers and non-developers alike, and that there is a visual way (as opposed to a programmer's) to do most things. Here is an example.

Test suites

Let's imagine that a programmer writes a bunch of scripts and wants them to run in a certain order and with a certain number of repetitions. That programmer might write some code that looked something like this:

```
function DoMainTest() {

  InitializeTestRun;
  DoUnitTesting();
  DoFunctionalTesting();

  for (var lIndex = 0; lIndex < 10; lIndex++) {
    CreateVirtualUsers();
    DoLoadTesting();
  }

  DoDataDrivenTesting();
  FinalizeTestRun();

}
```

For most programmers, this sort of code is not that daunting, but most non-programmers will hope there's a more visual way to do this. Come to think of it, there are some programmers who hope the same thing. Well, there is a visual way (see figure 4). Assuming that you've recorded some scripts (like DoUnitTesting and DoFunctionalTesting, etc.) you can use TestComplete's Test Suites feature to create the same test run visually. Figure 4 will result in the same test run as the script code above.

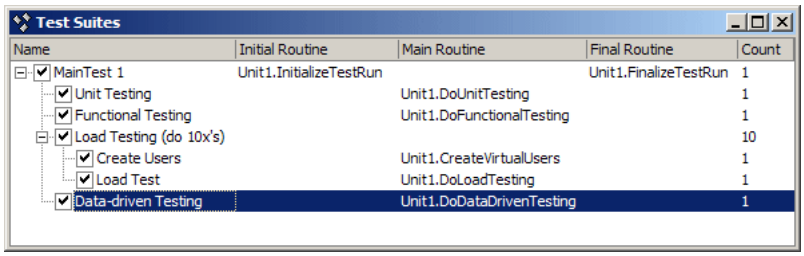

Figure 4 – *Test Suites. A visual (non-programmatic) way to build test runs.*

There are many other examples in TestComplete, one of the best being in the realm of object comparisons. Once you understand how object comparisons work in TestComplete, you can find an object, capture it, store it and have TestComplete generate comparison code for you all in just a few steps. The end result being that you are able to paste some generated code into your scripts, compare your objects in your test run and look really good without trying real hard.

White-box Testing

Transparency is an important concept in the testing arena. Transparency, in this case, is in terms of how much can be seen inside your tested application. No doubt you know that there is a lot of code operating down inside your application where no one can see it, but for testing thoroughness, it would be good to be able to examine what's going on in there. TestComplete has ways of seeing into an application; how far it can see is determined by how the application is compiled. For an application like Microsoft's MSPaint, TestComplete can see only the surface of the application – basically, it can see the information that Windows will disclose about the application. This is called "black-box testing" because you're basically testing a black-box that you can't see into. You can still poke at the box and measure it, but there are mysteries in there. Now, depending on how your applications are compiled, TestComplete can see farther into an application. Under optimal conditions, TestComplete can perform white-box testing (also called transparent-box testing), so called because there is nothing hidden inside the box (your application). The great benefit of white-box testing is that you can automate the external stimuli of "poking at the box" and then you can look down into the box to see what happened.

It's important to note that the level of transparency has no effect on whether or not TestComplete can perform automated testing on an application. You could conceivably create automated test projects for MSPaint and other black-box applications; you just couldn't look inside them to assess the internal state. The same is true of your applications. It's possible to still automate tests of your applications as black boxes.

The Results Log

All these testing capabilities are not much use unless the results can be displayed in a coherent manner. TestComplete really shines in this area. The TestComplete results log is a persistent (meaning that the results of past test runs stick around) and customizable record of the results of your tests stored in an XML format. This log can be viewed in TestComplete or, because it's in XML, it can be opened in an XML-supportive browser (only Internet Explorer at this time). Thus, the results can be viewed by people who do not have TestComplete on their computers. This is important because, depending on your accountability needs, you could post the results logs to a server and send out emails

containing the link to those who want to be informed.

Also, because the results are in XML, they can be processed by any tool which handles XML.

What follows are two screenshots of a typical result log. Figure 5 is the result log viewed in TestComplete. Figure 5a is the result log viewed in Internet Explorer.

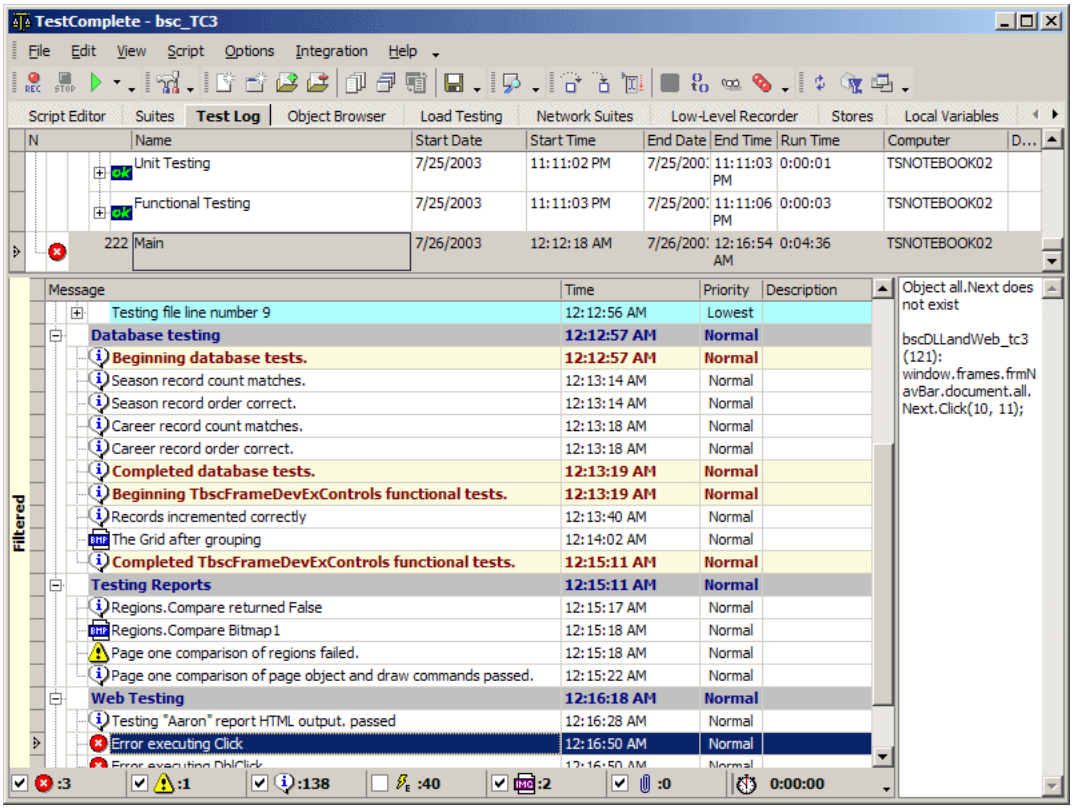

Figure 5 - *a result log viewed in TestComplete. Notice the customized colors, fonts and comments. Notice also the filter bar at the bottom allowing you to view only the messages you want.*

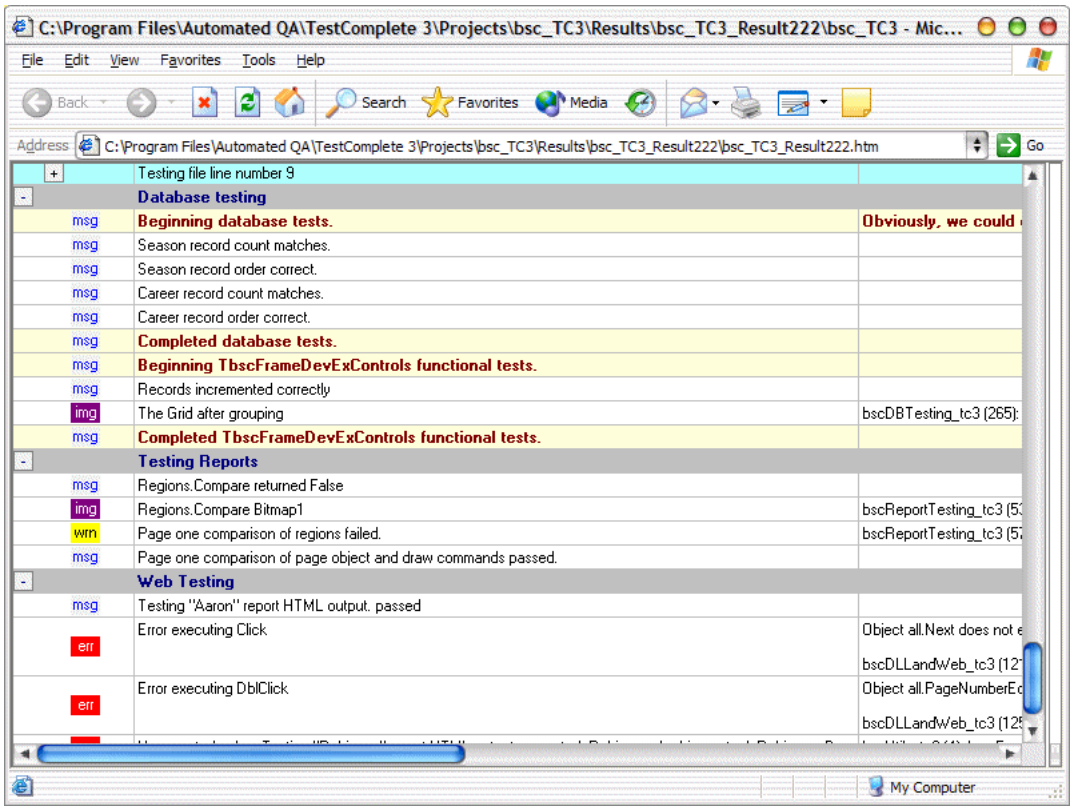

Figure 5a - *a result log viewed in Internet Explorer. Notice how the log is formatted to look just like it did when viewed in TestComplete.*

Conclusion

I hope that this paper will have answered some of your questions about testing in general and TestComplete in particular. TestComplete is a powerful test automation tool and is practical, and affordable for QA teams of any size. Please consider downloading the evaluation version of the tool and trying it out. You can also send any questions you might have to me at robertl@automatedqa.com.

Index