**Software Test Automation and the Product Life Cycle**

***Implementing software test in the product life cycle***

**The PLC and Automated Test**

A product's stages of development are referred to as the product life cycle (PLC). There is considerable work involved in getting a product through its PLC. Software testing at many companies has matured as lessons have been learned about the most effective test methodologies. Still, there is a great difference of opinion about the implementation and effectiveness of automated software testing and how it relates to the PLC.

Computers have taken over many functions in our society that were once "manual" operations. Factories use computers to control manufacturing equipment and have cut costs enormously. Electronics manufacturing use computers to test everything from microelectronics to circuit card assemblies. Since automation has been so successful in so many areas, does it make sense that a software program should be used to test another software program? This is referred to as "automated software testing" for the remainder of this article.

Software testing using an automatic test program will generally avoid the errors that humans make when they get tired after multiple repetitions. The test program won't skip any tests by mistake. The test program can also record the results of the test accurately. The results can be automatically fed into a database that may provide useful statistics on how well the software development process is going. On the other hand, software that is tested manually will be tested with a randomness that helps find bugs in more varied situations. Since a software program usually won't vary each time it is run, it may not find some bugs that manual testing will. Automated software testing is never a complete substitute for manual testing.

There has been plenty of debate about the usefulness of automatic software testing. Some companies are quite satisfied with the developer testing his/her own work. Testing your own work is generally thought of as risky since you'll be likely to overlook bugs that someone not so close to the code (and not so emotionally attached to it) will see easily. As soon as the developer says it's done they ship it. The other extreme is the company that has its own automatic software test group as well as a group that tests the software manually. Just because we have computers does that mean that it is cost effective to write tests to test software and then spend time and resources to maintain them? The answer is both yes and no. When properly implemented, automated software test can save a lot of time, time that will be needed as the software approaches shipping.

This is where the PLC comes in. How effectively you make use of the PLC will often be dependent on your programming resources and the length of the PLC. Companies large and small struggle with software testing and the PLC. Hopefully, this discussion of the PLC should help you determine when to use automation and when manual testing is preferred. This should help you answer the questions: "Why should I automate my software testing?" "How can I tell if automation is right for my product?" "When is the best time to develop my test software?".

**The Product Life Cycle**

As we discuss the use of automated and manual testing, we need to understand what happens in each phase of the product life cycle. The PLC is made up of six major stages, the Design Phase, the Code Complete Phase, the Alpha Phase, the Beta Phase, the Zero Defect Build Phase, and the Green Master Phase. You can think of the PLC as a timeline showing the development of a software product. These are the major milestones that make up the Product Life Cycle. Products that follow these guidelines for implementation of the PLC will have a much better chance of making it to market on time.

The implementation of the PLC varies widely from company to company. You can use this as a guide for future reference to assist you in your automation efforts. Your implementation will vary from the ideal PLC that is discussed here, but your software's success may depend on how well you've implemented its PLC. If your PLC is to include automated testing you should pay attention to which automated tasks are performed during each phase.

For each phase we'll describe it, define its special importance and discuss how to incorporate software automation into your project. Most other discussions of the PLC don't include the lessons learned about test automation. This should be your "one-stop" guide to help you know how and when automation fits into the PLC.

**Design Phase**

***What is the Design Phase?*** The design phase begins with an idea. Product managers, QA, and Development get together at this point to determine what will be included in the product. Planning is the essence of the design phase. Begin with the end in mind and with a functional specification. Write down all of your plans. What will your product do? What customer problems does it solve?

Incorrectly, some companies don't include Quality Assurance (QA) in the design phase. It is very important that QA be involved as early as possible. While developers are writing code, QA will be writing tests. Even though QA won't really have the total picture of the product, they will want to get as much of a jump on things as possible. Remember, that the primary purpose of QA is to report status. It is important to understand the product's status even early in the Design Phase.

***Why is the Design Phase important?*** If you think you're too short on time to write up a functional description of your product, then consider the extra time involved to add new features later on. Adding features later (especially once the Code Complete Phase has been reached) is known as "feature creep". Feature creep can be a very costly haphazard way to develop your product, and may materially interfere with delivery of the software.

Automation activity during the Design Phase. As soon as the functional specification is written, create all test cases so that they can be run manually. Yes, that's right, manually! These manual tests are step-by-step "pseudo" code that would allow anyone to run the test. The benefits of this approach are:

1. Your test cases can be created BEFORE ever seeing the software's user interface (UI). It is too soon to automate tests at this point in time, but you can create manual tests with only a small risk of changes that will occur . This is a point of great frustration for those who have tried to implement automated test scripts too early. Just as soon as the test script is written changes in the UI are bound to be introduced and all the work on the script is found to be for nothing.
2. When (not if) the code is modified, you will always have manual procedures that can be adapted to the change more quickly than an automated test script. This is a great way to guarantee that you will at least have tests you can perform even if automation turns out to not be feasible. (Note: one of the objections to software test automation is that the tests must be continually updated to reflect changes in the software. These justifiable objections usually stem from the fact that automation was started too early.)
3. Test methods can be thought out much more completely because you don't have to be concerned with the programming language of the automation tool. The learning curve of most automation tools may get in the way of writing meaningful tests.

If you have the resources available, have them begin training on the test tools that will be used. Some members of the team should start writing library routines that can be used by all the test engineers when the start their test coding. Some of these routines will consist of data collection/result reporting tools and other common functions.

After the manual test cases have been created decide with your manager which test cases should be automated. Use the Automation Checklist found later in this article to assist you in deciding what tests to automate. If you have enough manpower you may want to have an test plan team and an automation team. The test plan team would develop tests manually and the automation team would decide which of the manual tests should be run automatically (following the guidelines of the Automation Checklist later in this article). The automation team would be responsible for assuring that the test can be successfully and cost effectively automated.

Sometime during the design phase, as soon as the design is firm enough, you'll select the automation tools that you will need. You don't have to decide exactly which tests need to be automated yet, but you should have an idea of the kinds of tests that will be performed and the necessary capabilities of the tools. That determination is easier as the software gets closer to the code complete phase. Your budget and available resources will begin to come into play here.

For just a moment, let's discuss some of the considerations you should use in selecting the test tools you need. You'll also want to keep in mind the Automation checklist later in this column. It will help you determine if a test should be automated. There are a few good testing tools including Apple Computer's Virtual User (VU) (See the September, 1996 article "Software Testing With Virtual User", by Jeremy Vineyard) and Segue's QA Partner (Segue is pronounced "Seg-way").

Is there a lot of user interface (UI) to test? Software with a lot of UI is well suited for automated black box testing. However, some important considerations are in order here. You need to get involved with development early to make sure that the UI can be "seen" by the automation tool. For example: I've seen programs in which a Virtual User 'match' task (note: a task is what a command is called in Virtual User) couldn't find the text in a text edit field. In those cases, this occurred because the program didn't use standard Macintosh calls, but rather was based on custom libraries that provided UI features their own way.

Will the automated test environment effect the performance or operation of the system being tested? When you're trying to test the latest system software, you don't want the testing system changing the conditions of the test.

Is the speed that the tests run a consideration? If you're trying to measure the performance of a system you'll want to make sure that the conditions are as much like the "real world" as possible. You should consider the amount of network traffic that is present while you're running your tests. Also, the speed of your host processor can effect the time it takes your tests to run. You should schedule your tests so that you minimize the possibility of interfering with someone else on your network. Either isolate your network from others or warn them that you will be testing and that there is a possibility that their network activity may slow down.

What kinds of tests will be performed? The lower the level the testing is, the more likely white box testing should be used. A good example of this would be if you have a function that does a calculation based on specific inputs. A quick C program that calls the function would be much faster and could be written to check all the possible limits of the function. A tool like VU would only be able to access the function through the UI and would not be able to approach the amount of coverage that a C program could do in this situation.

Is there a library of common functions available or will you have to write them yourself? It will save a lot of time if you don't have to develop libraries yourself. No matter how extensive the command set, efficient use of library functions will be essential. Libraries that others have written may be useful; you can modify them to meet your own needs.

What will be the learning curve for a script programmer? The time it takes will depend greatly on the kind of testing you have to do and the experience of the programmer. If you've done your homework on the available test tools, you should know what to expect. Some companies even offer training courses (for a price) in their test software.

Can your automation tool automatically record actions for you? Some tools do this, but don't expect to rely on this too heavily. Tools that I've seen that do this end up creating code that has hard coded strings and tend to be organized in a sequential manner rather than by calling procedures. These recorded scripts are harder to maintain and reuse later. If you plan to use the same script for international testing, modifying the script will mean much more work. If you want to record actions, I recommend that you do it only to create short functions and you should edit the script after recording to remove the unwanted hard coded strings, etc.

Can you run AppleScript scripts from the tool's script language? This is a very useful feature since AppleScript scripts are so easy to write and can add additional functionality to your test tool.

In preparing this article, I encountered several "pearls" worth relating:

"Success in test automation requires careful planning and design work, and it's not a universal solution. ... automated testing should not be considered a replacement for hand testing, but rather as an enhancement." (Software Testing with Visual Test 4.0, forward by James Bach, pg. vii)

"The quality assurance engineers then come on the scene... and begin designing their overall test plan for the features of the product...."

"The goal is to have the test plan and checklists laid out and ready to be manually stepped through by the test engineers when each feature is completed by the programmers. Each item on a checklist is considered a scenario and related scenarios are grouped into test cases." (*Software Testing with Visual Test 4.0*, pg. 5-6)

**Code Complete Phase**

***What is the Code Complete Phase?*** At this major milestone the code has been completed. The code has been written, but not necessarily yet debugged. (Development may try to claim they are at code complete even though they may still have major coding still left to do. Go ahead and let them declare the code complete, but don't let them get to Alpha until the code really is completely written.)

***Why is the Code Complete Phase important?*** Sooner or later you'll have to get to a point where new code is no longer being written, and the major effort is in fixing bugs. Development will be relieved to get to this point as now they don't have to be as concerned with the initial coding and can concentrate on refining the existing product. (This is why they will try to claim they are at code complete even when they are not).

Automation activity during the Code Complete Phase Although the UI may still change, QA can begin writing Automatic test cases. The tests that should be written at this point are breadth tests that tell the status of the overall software product. Don't write tests which stress the product until you get close to Alpha. The product will probably break very easily. Some acceptance (or "smoke") tests should also be created to give a quick evaluation of the status of a particular build. Before reaching the Alpha phase there should also be tests written to test the Installer, boundary (or stress tests), compatibility (hardware and OS), performance, and interoperability.

Somewhere just before code complete, you will need to decide which tests should be made into automatic tests and what test tools to use. Use the following checklist to help you determine which tests should be automated:

**Automation Checklist**

If you answer yes to any of these questions, then your test should be seriously considered for automation.

**Can the test sequence of actions be defined?**

Is it useful to repeat the sequence of actions many times? Examples of this would be Acceptance tests, Compatibility tests, Performance tests, and regression tests.

**Is it necessary to repeat the sequence of actions many times?** (See *Testing Computer Software*, pg. 196 and 282).

**Is it possible to automate the sequence of actions?** This may determine that automation is not suitable for this sequence of actions.

**Is it possible to "semi-automate" a test?** Automating portions of a test can speed up test execution time.

**Is the behavior of the software under test the same with automation as without?**This is an important concern for performance testing.

**Are you testing non-UI aspects of the program?** Almost all non-UI functions can and should be automated tests.

**Do you need to run the same tests on multiple hardware configurations?** Run ad hoc tests (Note: Ideally every bug should have an associated test case. Ad hoc tests are best done manually. You should try to imagine yourself in real world situations and use your software as your customer would. As bugs are found during ad hoc testing, new test cases should be created so that they can be reproduced easily and so that regression tests can be performed when you get to the Zero Bug Build phase.) An ad hoc test is a test that is performed manually where the tester attempts to simulate real world use of the software product. It is when running ad hoc testing that the most bugs will be found. It should be stressed that automation cannot ever be a substitute for manual testing.

**Alpha Phase**

***What is the Alpha Phase?*** Alpha marks the point in time when Development and QA consider the product stable and completed. The Alpha Phase is your last chance to find and fix any remaining problems in the software. The software will go from basically functional to a finely tuned product during this phase.

***Why is the Alpha Phase important?*** Alpha marks a great accomplishment in the development cycle. The code is stable and the most major bugs have been found and fixed.

**Automation Activity During The Alpha Phase**

At this point you have done the tasks that need to be done in order to reach Alpha. That is, you have all your compatibility, interoperability, and performance tests completed and automated as far as possible. During Alpha you'll be running breadth tests every build. Also you'll run the compatibility, interoperability, and performance tests at least once before reaching the next milestone (beta). After the breadth tests are run each build, you'll want to do ad hoc testing as much as possible. As above, every bug should be associated with a test case to reproduce the problem.

**Beta Phase**

***What is the Beta Phase?*** The product is considered "mostly" bug free at this point. This means that all major bugs have been found. There should only be a few non essential bugs left to fix. These should be bugs that the user will find annoying or bugs that pose relatively no risk to fix. If any major bugs are found at this point, there will almost definitely be a slip in the shipping schedule.

**Automation activity during the Beta Phase**

There's no more time left to develop new tests. You'll run all of your acceptance tests as quickly as possible and spend the remaining time on ad hoc testing. You'll also run compatibility, performance, interoperability and installer tests once during the beta phase.

Remember that as you do ad hoc testing every bug should have an associated test case. As bugs are found during ad hoc testing, new test cases should be created so that they can be reproduced easily and so that regression tests can be performed when we get to the Zero Bug Build phase.

**Zero Defect Build Phase**

***What is the Zero Defect Build Phase?*** This is a period of stability where no new serious defects are discovered. The product is very stable now and nearly ready to ship.

**Automation Activity During The Zero Defect Build Phase**

Run regression tests. Regression testing means running through your fixed defects again and verify that they are still fixed. Planning for regression testing early will save a lot of time during this phase and the Green Master phase.

**Green Master**

What is the Green Master Phase? Green Master is sometimes referred to as the Golden Master or the final candidate. The product goes through a final checkout before it is shipped (sent to manufacturing).

**Automation activity during the Green Master Phase**

After running general acceptance tests, run regression tests. You should run through your fixed defects once again to verify that they are still fixed. Planning for regression testing early will save a lot of time during this phase.

**Understanding the PLC will help you select your automation tools**

Perhaps this review of the PLC is 'old hat' for you. In my experience, reviewing the process usually helps to focus the project. Since software test has been evolving over the years, and many companies have been struggling with how to implement it, we can always use all the help and advice that we can get. There are some good sources of information that will help. Take a look at the Software Testing Laboratories web site at URL [http://www.stlabs.com](http://www.stlabs.com/). There are a few relevant articles in James Bach's archives at<http://www.stlabs.com/LABNOTE.HTM>. The Software Testing Labs is mostly geared toward MS Visual Test, but the QA principles involved are the same for any platform, Macintosh included.

This review of the PLC represents an ideal situation where Development and QA both buy into this same way of doing the software business. It will also require some commitment on the part of Management to assure that each phase is supported and accepted. Once all of those conditions, everyone involved needs to be focused on making sure that no short cuts are taken around the agreed upon PLC. Hopefully this discussion has been helpful to you in automating your software tests.

**Glossary**

**Word or phrase**

Definition.

**Ad Hoc Testing**

Goal oriented passing through the product. Sometimes to prove or disprove a notion of how the product will behave.

**Alpha Test**

The part of the Test Phase of the PLC where code is complete and the product has achieved a degree of stability. The product is fully testable (determined by QA). All functionality has been implemented and QA has finished the implementation of the test plans/cases. Ideally, this when development feels the product is ready to be shipped.

**Automated Testing**

Creation of individual tests created to run without direct tester intervention.

**Beta Test**

The part of the Test Phase of the PLC where integration testing plans are finished, depth testing coverage goals met; Ideally, QA says product is ready to ship. The product is stable enough for external testing (determined by QA).

**Black Box Test**

Tests in which the software under test is treated as a black box. You can't "see" into it. The test provides inputs and responds to outputs without considering how the software works.

**Boundary Testing**

Test which focus on the boundary or limit conditions of the software being tested. (Some of these tests are stress tests).

**Breadth Testing**

Matrix tests which generally cover all product components and functions on an individual basis. These are usually the first automated tests available after the functional specifications have been completed and test plans have been drafted.

**Breath Testing**

Generally a good thing to do after eating garlic and before going out into public. Or you may have to take a breath test if you're DUI.

**Bug**

A phenomenon with an understanding of why it happened.

**Code Complete**

Phase of the PLC where functionality is coded in entirety; bug fixes are all that are left. All functions found in the Functional Specifications have been implemented.

**Code Freeze**

When development has finished all new functional code. This is when development is in a "bug fixing" stage.

**Coding Phase**

Phase of the PLC where development is coding product to meet Functional/Architectural Specifications. QA develops test tools and test cases during this phase.

**Compatibility Test**

Tests that check for compatibility of other software or hardware with the software being tested.

**Concept Phase**

Phase of the PLC where an idea for a new product is developed and a preliminary definition of the product is established. Research plans should be put in place and an initial analysis of the competition should be completed. The main goal of this phase is to determine product viability and obtain funding for further research.

**Coverage analysis**

Shows which functions (i.e., GUI and C code level) have been touched and which have not.

**Data Validation**

Verification of data to assure that it is still correct.

**Debug**

To search for and eliminate malfunctioning elements or errors in the software.

**Definition Phase**

See Design Phase.

**Dependency**

This is when a component of a product is dependent on an outside group. The delivery of the product or the reaching a certain milestone is affected.

**Depth Testing**

Encompasses Integration testing, real world testing, combinatorial testing, Interoperability and compatibility testing.

**Design Phase**

Phase of the PLC where functions of the product are written down. Features and requirements are defined in this phase. Each department develops their departments' plan and resource requirements for the product during this phase.

**Dot Release**

A major update to a product.

**Feature**

A bug that no one wants to admit to.

**Focus**

The center of interest or activity. In software, focus refers to the area of the screen where the insertion point is active.

**Functional**

Phase of the PLC defining modules.

**Specifications**

Their implementation requirements and approach, and exposed API. Each function is specified here. This includes the expected results of each function.

**GM**

See Green Master.

**Green Master (GM)**

Phase of the PLC where the certification stage begins. All bugs, regressed against the product, must pass. Every build is a release candidate (determined by development).

**GUI**

Graphical User Interface.

**Inline**

Phase of the PLC after shipping (STM) where bugs are fixed for interim release. Maintenance of the product involves cleaning up bugs that are found after STM. Inlines are created to address these problems.

**Integration Testing**

Depth testing which covers groups of functions at the subsystem level.

**Interoperability Test**

Tests that verify operability between software and hardware.

**Load Test**

Load tests study the behavior of the program when it is working at its limits. Types of load tests are Volume tests, Stress tests, and Storage tests.

**Localization**

This term refers to making software specifically designed for a specific locality.

**Maintenance Release**

See Inline.

**Metrics**

A standard of measurement. Software metrics are the statistics describing the structure or content of a program. A metric should be a real objective measurement of something such as number of bugs per lines of code.

**Milestones**

Events in the Product Life Cycle which define particular goals.

**Performance Test**

Test that measures how long it takes to do a function.

**Phenomenon**

A flaw without an understanding.

**PLC**

Product Life Cycle - see Software Product Life Cycle.

**Pre-Alpha**

Pre-build 1; product definition phase. (Functional Specification may still be in process of being created).

**Product Life Cycle**

The stages a product goes through.

**(PLC)**

from conception to completion. Phases of product development includes: Definition Phase, Functional/Architectural Specification Phase, Coding Phase, Code Complete Phase, Alpha, Beta, Zero Bug Build Phase, Green Master Phase, STM, and Maintenance/Inline Phase.

**Proposal Phase**

Phase of the PLC where the product must be defined with a prioritized feature list and system and compatibility requirements.

**QA Plan**

A general test plan given at the macro level which defines the activities of the test team through the stages of the Product Life Cycle.

**Real World Testing**

Integration testing which attempt to create environments which mirror how the product will be used in the "real world".

**Regression Testing**

Retesting bugs in the system which had been identified as fixed, usually starting from Alpha on.

**Resource**

People, software, hardware, tools, etc. that have unique qualities and talents that can be utilized for a purpose.

**Risk**

Something that could potentially contribute to failing to reach a milestone.

**STM**

See Ship to Manufacturing.

**Storage Tests**

Test how memory and space is used by the program, either in resident memory or on disk.

**Stress Test**

Tests the program's response to peak activity conditions.

**Syncopated Test**

A test that works in harmony with other tests. The timing is such that both tests work together, but yet independently.

**Test Case**

A breakdown of each functional area into an individual test. These can be automated or done manually.

**Test Phase**

Phase of the PLC where the entire product is tested, both internally and externally. Alpha and Beta Tests occur during this phase.

**Test Plan**

A specific plan that breakdown testing approaches on a functional area basis.

**Test Suite**

A set of test cases.

**Usability**

The degree to which the intended target users can accomplish their intended goals.

**Volume Tests**

Test the largest tasks a program can deal with.

**White Box Test**

It is used to test areas that cannot be reached from a black box level. (Sometimes called Glass Box testing).

**Zero Bug Build**

Phase of the PLC where the product has stabilized in terms of bugs found and fixed. Development is fixing bugs as fast as they are found, the net resulting in zero bugs on a daily basis. This is usually determined when after a few builds have passed. This is the preliminary stage before Green Master.