*IOSR Journal of Computer Engineering (IOSR-JCE)*
*e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 20, Issue 4, Ver. I (Jul - Aug 2018), PP 34-39*
*www.iosrjournals.org*

# Computerized Testing of Simulation and Embedded Software

Dr. Amamer Khalil Masoud Ahmidat, Muhamad Abdulla Muhamad Abdussalam
*Higher Institute of Medical Technology in Baniwaleed, Libya*
*Higher Institute of Medical Technology ,Baniwalid, Libya*
*Corresponding Author: Dr. Amamer Khalil Masoud Ahmidat*

***Abstract:*** *This paper deals a system for computerized testing of simulation and embedded software. This test makes thorough, frequent testing easy, which means errors will be caught more quickly. Computerized testing also means that more of the engineer's time can be spent on development, and that development can continue closer to field tests. In the proposed system, source code check-in triggers a chain of tests. These tests would include static checks, compilation, and test execution. For embedded software, it proposes extending the computerized testing to execution on digital hardware set aside for testing purposes*.

Date of Submission: 02-07-2018 Date of acceptance: 21-07-2018

## I. Introduction

Software testing is a broad term, but in this paper we will concentrate on two types of testing that are automatable: test by execution, and static analysis.

The optimist tests software in order to check that it satisfies expectations (that is, specifications); in this view, "A bug is a test case you haven't written yet". Combinatorial explosion in input space leads the pessimist to Dijkstra's view that "Program testing can be used to show the presence of bugs, but never to show their absence!". This is true, but software testing remains a practical means to

i. Verify an implementation against specification, however superficially;
ii. Allow implementation changes; if a test (or suite thereof) passes before and after optimizing an algorithm implementation, there is a good chance that the change is successful.
iii. Capture development knowledge, for instance if the test suite is expanded every time a bug is found.

Software testing does not mean that other assessment techniques, such as code review, are unnecessary. It is unfortunately easy to write code that passes tests and follows naming conventions,
etc., but is still difficult to read and maintain. Test automation means reviewers don't have to deal with mundane problems such as naming conventions, and can focus on how clearly implementation expresses intent.

Something separating many types of software testing from code review is the possibility of automation; that is, tests can be run, without human initiation, any time a software change is made. Automation relieves the developer of the responsibility of test execution, which means they can be run more often, and can be more extensive; automation also allows repeatability.

All of this comes at a cost, namely in developing the tests themselves, and developing and maintaining the automation infrastructure. These costs can be reduced by using existing test libraries (e.g., Google Test (Google)) and test managers (e.g., Jenkins (Kohsuke Kawaguchi)).

The cynic might note that this all amounts to testing one complicated set of software with another, equally complicated set of software, presumably doubling the total number of bugs.

This criticism has merit; the benefits of this approach will not be reaped if such a project is initiated at the end of development; in this case the software to be tested already exists, will have been tested to some extent, and engineers will probably spend more time fixing the tests than the product.

However, if a computerized test system is set up at the beginning of the project, once it is working engineers can focus on product implementation. In this case tests are developed with the product, and continue to be used throughout the development period. As field test deadlines approach, engineers don't need to budget as much time for software testing, since tests already exist and are being executed all the time.

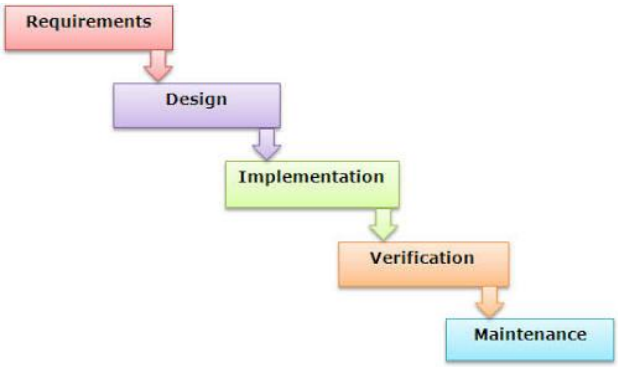
## II. Methods Of Testings

**Software testing levels**

The typical software testing levels and how they relate to the system requirements, specification and design, are as follows:

- Acceptance testing
- System testing
- Integration testing
- Unit testing
- System requirements
- Unit design
- Development
- System design
- System specification
- Conformance testing
- Development plan

Integration testing, informal testing used to verify interfaces and the interactions between software units. This testing might initially be down on a development system, but should finally be done on the embedded system. System testing, formal testing used to verify that a system meets specifications. These tests are usually executed according to test instructions that are coupled to specifications. These tests would be conducted in the embedded system, possibly using a HILS as a test bench. Acceptance testing, formal testing used to verify a system against its requirements. This sort of testing could be conducted in the field, and here the software forms part of the total system under test.



**Software testing tools**

**Static checkers**

Static checkers evaluate source code; in these tests, the software is not executed. The most basic such tool is the compiler: source code must be compliable in order to be executed; besides this, compilers give warnings for code that may be buggy. Static checkers can check conformance to coding standard rules (e.g., "every else-if chain must end with a bare else"), code metrics and code style (e.g., whitespace checkers).

**Target emulators**

While it is useful to check implementations on development systems (typically Intel x86-class Windows systems), they must be tested on the final embedded target. A useful in-between step is to use a target platform emulator; this allows for testing of object code compiled for the final target on a development system.

**Code coverage**

Code coverage tools let you check which parts of the product source code are executed during testing. This is usually fairly straightforward for code executed natively on development systems, but not always possible for embedded systems. Microsoft Visual Studio (Microsoft) and gcc (Free Software Foundation, Inc.), both in widespread use, have code coverage capabilities.

**Test manager**

A test manager is a system that automates test execution; typically such a system will check source out of a repository, compile it, and execute a set of tests. This chain of events can be triggered by a repository check-in, or be done periodically. The manager can record the test binaries and results, and notify engineers in the event of failure.

Jenkins (Kohsuke Kawaguchi) is an open-source test management system.The Apache software foundation's Jenkins CI server interface, which contains a list of test names and the status of each test.

**Test support libraries**

Test support libraries make it easier to write tests; they allow for organization of test suites, and for test conditions to be concisely and clearly expressed. Many such libraries are in socalled "xUnit" family, and variants exist for C, C++, C#, Java, Matlab, and Python.

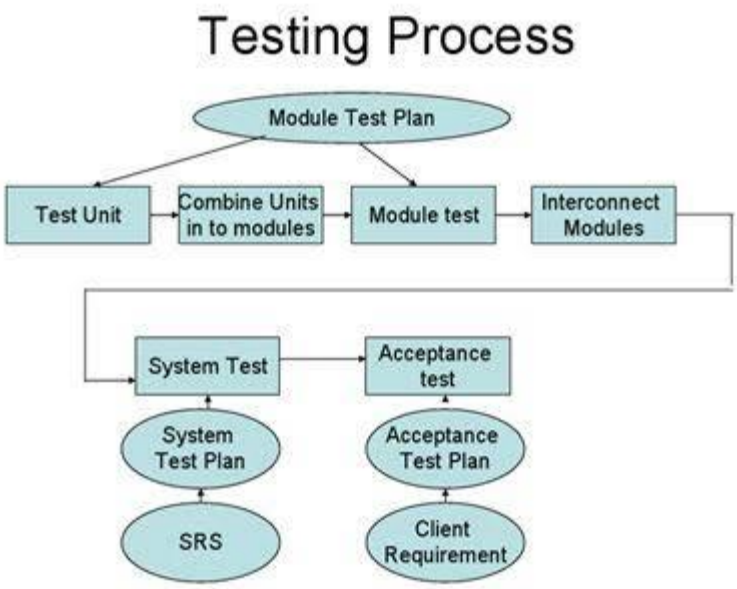
**Software-only testing**

The software-only test environment tests the unit/system on a standard computer. It is flexible, since it can be used across most testing levels, from unit testing to system testing. This is typically an easily controlled environment, where the tests are repeatable.

The test manager checks out the updated source code from the repository, and follow:

- Compiles the new algorithm and tests for Windows x64 without warning or error.
- Runs the tests; the newly committed tests pass, but an older test fails because it assumed sensor offset errors would not be calibrated out.
- Code coverage analysis from the test execution reveal that the error-handling in the new code was not tested.
- The code is then compiled for an emulated ARM target environment; several warnings are signaled by the cross-compiler.
- The unit tests are run in an ARM emulator. Some of the new tests fail due to subtle differences in floating-point between the Intel and ARM math libraries.
- A static analyzer detects that several new variables do not follow the project's naming conventions.
- The commit is marked as failing; the test manager opens a new issue on the tracking system, and sends an e-mail with a summary of the problems found to the erring developer.
- Armed with this knowledge, she fixes the problems and makes a new commit that passes the software-only test suite.

The example illustrates that the developer must be able to execute the various tests outside of the test automation framework. If the testing queue is full, turnaround time for test execution will be long, and a developer might have started new work based on a substandard baseline by the time failure notification is sent.




**Stand-alone embedded testing**
Telemetry recorder
System under test
Power supply
Revision control server
Continuous integration server
Telemetry analyser
Workstation
<<controls>>
<<controls>>
<<triggers>>
<<powers>>
<<feeds>>
<<reads>> <<reports>>
<<emulates>>
<<commands>>
<<responds>>
<<reports>>
Issue tracking
server
Emulator
Test controller
<<controls>>
<<reports>>
<<feeds>>
<<reports>>

The embedded test environment tests the stand-alone embedded system. It requires, at a minimum, a way to power the embedded system and also a way to communicate with it, e.g., an
ARCNET (ARCNET Trade Association) interface. This is typically a controlled environment, where the test scenarios are repeatable.
Above list shows the computerized embedded test environment components:

- All the components described in Figure 3
- Power supply: powers the System Under Test (SUT).
- Emulator: emulates code execution on the SUT, without booting from non-volatile memory.
- Telemetry recorder: records SUT telemetry.

- Telemetry analyser: analyses recorded SUT telemetry to determine if test passes/fails.

Continuing the example from the previous section: the second commit passed the softwareonly tests, and so the test manager triggers the stand-alone embedded testing via a dedicated embedded tester.
1. The latest embedded source code is checked out, and cross-compiled warning-free into a test application to be loaded onto a single-processor ARM system.
2. The embedded tester powers on an attached ARM board, and uses this board's boot-loader to load the new application.
3. The tester executes the stand-alone test suite: via a telemetry network, it sends test inputs to the ARM test application, and records the resulting outputs.
4. These outputs are compared to references, and are errors are within tolerance specifications in all cases. However, the algorithm execution time sometimes exceeds its time limit.
5. The commit is marked as failing. Another issue is opened, and an e-mail with details of the timing problem are sent to the developer. She commits a modified implementation that caches an often-used matrix multiplication result; this commit successfully pass through the battery of software-only tests, and is fast enough to pass the stand-alone embedded test.

Similarly to the software-only only example, this illustrates the need for the developer to run stand-alone embedded tests outside of the test automation framework. If test hardware is limited, this might mean stalling the test queue during debugging.

**Operational testing**

This test environment tests the completely integrated system in its expected operational environment, i.e. field testing. Problems found at this level could be very costly to resolve, since the test could be aborted in the event of a major problem, or the system could be damaged beyond repair. The uncontrollable nature of this environment (and the logistics involved with it) means that its testing process is typically not repeatable, and computerized testing is infeasible.

## III. Conclusion

Computerized testing relieves developers and reviewers from the burden of routine testing. If used from early on in development, it allows for frequent, thorough testing; it can capture development history in the form of regression tests; and it can allow development to continue closer to deadlines.

Test automation can be used with unit test execution, including coverage checking, and static analysis. Tools exist to allow automation, to develop test suites, and to perform analysis of code and results.

Testing can be extended from typical "software-only" testing to executing in test frameworks on target embedded hardware. A simple example illustrates that developers also need to be able to execute tests outside the automation framework.

## References

[1]. Microsoft. Microsoft Visual Studio. http://www.visualstudio.com/.
[2]. Scientific Toolworks, Inc. Understand. https://scitools.com/.
[3]. Sysprogs UG. Prebuilt GNU toolchain for powerpc-eabi. http://gnutoolchains.com/ powerpc-eabi/.
[4]. Wiegers, Karl E. 2003. Software requirements. 2nd edition. Microsoft Press.
[5]. Wikipedia. xUnit—Wikipedia, the free encyclopedia. [Accessed 14-October-2014]. http:
[6]. Gimpel Software. PC-lint for C/C++. http://www.gimpel.com/html/pcl.htm.
[7]. Google. Google C++ testing framework. https://code.google.com/p/googletest/.
[8]. Jean-Philippe Lang. Redmine. http://www.redmine.org/.
[9]. Kohsuke Kawaguchi. Jenkins CI. http://jenkins-ci.org/.
[10]. Meyer, B. 2008. Seven principles of software testing. Computer 41 (8): 99–101. ISSN: 0018- 9162, doi:10.1109/MC.2008.306.

**Amamer Khalil Masoud Ahmidat** PhD in the computer and information Faculaty of electrical Engand information technical University of Kosice Slovaki (2003-2008) . BSc electronic .Eng Computer Dept (1987-1991).MSc in Computer engineering and information Faculaty of electrical Eng.and information technical University of Kosice Slovaki (1995-1998) . **Head of Higher Institute of Medical Technology in Baniwaleed-Libya,** Assistant Professor from ( 01-10-2013.)

**Muhamad Abdulla Muhamad Abdussalam** M.Sc. in Computer

Engineering Technical University of Kosice ,Slovak Republic. Bachelor of Science B.Sc. in Electronic Engineering, College of Electronic Technology Baniwalid. **Head of Study And Examinations Department, Higher Institute of Medical Technology ,Baniwalid, Libya**.

IOSR Journal of Computer Engineering (IOSR-JCE) is UGC approved Journal with Sl. No. 5019, Journal no. 49102.

* Dr. Amamer Khalil Masoud Ahmidat. " Computerized Testing of Simulation And Embedded Software ." IOSR Journal of Computer Engineering (IOSR-JCE) 20.4 (2018): 34-39.