Turkish Journal of Computer and Mathematics Education Vol.12 No.13 (2021), 1456-1465

Research Article

The Role of Programming Languages in Software Testing

Dr. Ojekudo, Nathaniel Akpofure
Department of Computer Science
Ignatius Ajuru University of Education
Port Harcourt
Rivers State
Nigeria
nathojekudo@gmail.com, nath.ojekudo@iaue.edu.ng

Fiberesima, Alalibo Ralph
School of Post Graduate Studies
Department of Computer Science
Faculty of Natural Science and Applied Sciences
Ignatius Ajuru University of Education Rumuolumeni
Port Harcourt
Rivers State
Nigeria
fiberesimaalaliboralph@gmail.com

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 4 June 2021

Abstract

Software testing can be described as the process of manually or automatically subjecting a component of code or a piece of software. Testing can be done statically, dynamically, passively, using the "Box" approach or using the levels testing methodology. It is obvious that in the world of software engineering, programming languages have a role to play in proving the validity of software through tests. However, these tests are also a challenge for programming languages as they often lack the tools to automate them beyond Unit tests and so are restricted to Quality Assurance manuals such as the Ministry of Defense handbook for software quality. This report shows how the problem of testing can be addressed in a programming language using Integrated Development Environment tools as well solving the problem of automated testing, and validates the need for suitability as a criterion for assessment of a language for fitness of purpose when programming for a particular industry. The aim of this report is to prove the need for agile methodology in software testing and also to stratify agile methods by creating a formal framework. To achieve this, the objectives are to develop a non-trivial piece of software, prove the software for validity using the programming language's automated testing methods, and validate using classic testing to eliminate any flaws found during automated testing. The methodology chosen is the agile methodology for creating the software application and analysis of documentation is carried out as part of the software development process. Implementation is done using the Integrated Development Environment for the C sharp (C#) programming language and tests are carried out using the white and black box strategy. The report is important because it uses experiments to show the limit of white and black box testing only of agile projects and argues for a continuation of a combination of methods from developer requirements, business requirements, unit tests and classic programming.

Introduction

Software can be described as the set of instructions and ancillary operating system codes and data used by a computer. Testing can be described as the process of using or trying something to see if it works, is suitable or obeys the rules. (Cambridge, n.d). Therefore, software testing can be described as the process of manually or automatically subjecting a component of code or a piece of software.

An example is when you go to a showroom to purchase a vehicle, the dealer does not just sell you the car or truck, he allows you to drive it about, at least around the driveway or on a nearby road. This is an example of testing. Another

example is when you purchase a television set, you are allowed to observe a series of channels being charged on the television set and possibly a flash drive may be plugged into the television to play movies while you observe. So with these examples, we can introduce software testing as the usage of a piece of software before accepting it by passing it through a series of tests to check for anomalies, defects or errors. The ability for software to be tested is one of the factors that influences a software to be known as good software. Therefore, once a software developer intends to produce an artifact, it is mandatory that that artifact is testable. Indeed, there are many tests that such a software developer must subject the artifact to, and we shall look at these in subsequent parts of the report.

So just as software can be described as the code or instructions that are executable on hardware or software to perform a specific task, and testing can be described as the act of subjecting a thing to a set of procedures to find out it's suitability or fitness of purpose for a task, software testing integrates both descriptions as the subjection of a code suite of instructions to various tests to find out it's suitability for various functionality, which range from individual tests to group tests to suites of tests.

Programming languages are primarily languages used to develop software, either operating system software or application software. The role of programming languages in development of various tests cannot be emphasized enough as apart from development, the code must also be tested. So there is a great interest in the development of software testing tools using the same programming languages used in developing them.

Another example is the intrinsic wrong nature of testing a car tire with a pressure gauge meant for a truck, or testing a car horn for the same range as a train whistle. Such tests are superfluous and they result in developing wrong parameters and lead to obviously wrong conclusion. All tests should be done on equivalent basis for equality in programming language to test so that parity can be achieved. Another technical reason behind testing is that there is often an impedance between what the developer wants and the desire of the customer or funding entity. These tests reassure the customer that what is expected is the correct outcome.

Developer outcomes are often embedded in the Software Requirements Specification document and business requirements in the Business Requirements Specification document, and both are checked manually to verify that they meet the same exact specific objectives before they are signed off. The signing of on the Business Requirements Specification document may include features like Look and Feel of the software, response time, use of shortcut keys and user interaction, while Software Requirements Specification document objectives will include methods and the correct output, integration of various methods, white box testing of all the instructions and black box testing of the instructions. The software may be even integrated with compatible hardware to fulfill some aspect of the test suite.

In the past testing was relegated to the background and developer skills without testing skills still allowed a project to go from start to completion, but with the advent of Agile methodology in software development, it is now of the utmost importance that testing skills be added to the repertoire of skills possessed by software developers to properly execute an acceptable piece of software.

Testing is of various types and can be automated or manual. Manual testing involves the visual inspection of the output for adherence to the service contract of the software business requirements on a method by method basis by Quality Assurance members of the development team while Automated testing is the usage of specially developed tools in a programming language to test code written in that programming language. These tests range from small tests to test the output of a method to testing the entire system, which in cases of web frameworks may include tests of the hosting server and databases.

The problems generally observed in programming languages with respect to their role in software development include the need for correctness, or the independent, expert verification of the language for fitness of programming a particular application, the role of standardization or the ability of compilers and interpreters to be POSIX compliant and enable true cross platform interoperability between applications for acceptance testing and the absence of static analytic tools which provide code analysis that is more detailed than existing style check analyzers, proofreaders and indentation error highlighters.

Literature Review

In his work, Principles of the Agile Manifesto, Beck et al. (2001) stated the following principles behind the Agile Manifesto

He said the following principles are to be followed in developing software that meets the Agile standards:

1. The customer is to be seen as the highest priority in the software development process akin to he who pays the piper dictates the tune Software was to be built as quickly as possible and iterations made for improvement as quickly as possible.
2. In the Agile process the customer's right to a competitive edge over his business rivals is sacrosanct, and to maintain that edge he has the right to request any number of changes to be made to the software. The developer should welcome them and even help him think of innovations to the software
3. A shorter timescale is the best timescale as the software, which must work even at prototype level must be delivered at most weeks into project inception.
4. The need for the aforementioned requirements necessitates the frequency of meeting between customer and developer to be daily at times
5. Software projects must be assigned to developers who are motivated to do their best and for this to occur, developers must be given remunerative incentive to finish projects as fast as possible. They should also be given a free hand on the job to get the best possible quality of work done.
6. Project workspaces should place emphasis on face to face communication and as much red tape to higher management as is possible should be cut out
7. Only progress should be rewarded and this is done by assessment of the software's fitness of purpose.
8. All contributors to the software process, technical, financial and managerial should have equality of status provided there is equality in contribution.
9. Technical roles are not to be given subservient status and a technical manager should be the lead manager in any Agile project.
10. It should be easy to know by a few charts or conversation the amount of work done and amount of work left to be done.
11. A project team should be allowed to choose it's technical and other roles and those may come from within or without the organization, depending on the task.
12. All teams should adjust for efficiency periodically, and change their behavior accordingly for maximum gains per project.

These principles emphasize the importance of software testing as the aforementioned goals cannot be met in the absence of detailed and meticulous testing of software artefacts generated as a result of the software development life cycle.

Kaner (2006) has described software testing as information that those with a stake in a software project require on a procedural basis to ascertain the quality of the software being delivered. He further stated that this allows businesses contrast the process with their business requirements and use cases and appreciate the need for possible changes in business requirements, scope of work and time frame (Kaner et al., 1999).

There are different ways to approach the testing of software. A programmer can use testing approach, box approach or standards approach.

In the testing approach, testing can be differentiated according to whether the testing is done statically, dynamically or passively (Graham et al., 2008). In the static approach to testing, Integrated Development Environments (IDEs) have inbuilt checks for wrong syntax and wrong code arrangement as part of their environment build for a particular language. Oberkampf and Roy(2010) also explained that statically, testing is also complemented by manual code proofreading by programmers skilled in the detection of such flaws. A dynamically favored approach to testing a program meant for execution on the actual execution of the program. This allows for an examination of the output which returns true or false in the case of unit tests of pass or failure of execution in the case of larger tests. A good programming project should allow for output from execution to be saved to log files and the examination of these files lies in the domain of passive testing, Passively, due to non-interaction directly with the code and execution, but rather with the log files from the program execution, the programmers can verify the correct execution of the software and make decisions accordingly.

Another means of testing code for proper source code, flow and execution is the use of the "Box" methodology. The box methodology divides software into boxes based on the method or approach to the testing of the code. A "white" approach to testing software takes cognizance of the methods, classes, packages and modules in the piece of software and checks their interactions. It deliberately tests each unit of software by first verification of its validity of execution, next it tests the fault detection mechanism of each piece of software by deliberately making errors in the pieces of code and the entire system and finally it tests for faults in the interactions between units of code that make up the entire software (Limaye, 2009). A "black" approach to testing software examines the piece of software as a whole and does not seek to know anything about its internal workings. As a result, testing is done using classic programming methodology. This includes the setting of a boundary value for possible output, the construction of a Truth table to ascertain the validity of variables and their interactions in the program, the construction of state and transition charts to check the output variation as the code is executed and the supply of true and false values to the program to check the validity of the specifications underpinning the program (Black, 2011).

Finally, there is the "levels" at which testing can be achieved within a software development lifecycle phase and this includes the basic and system view of the piece of software. The Unit test is a functional test written by software programmers to have an insight into workings of the code in a "white box" format. Since the basic unit of every piece of software is the encapsulating mechanism for its methods and data, which in most programming languages is called a "class", the Unit test has as its aim the exposure of the workings for correctness of all the methods and class variables present in the class. This is done primarily by the injection of code that tests all the object initializers (constructors) and de-initializers (destructors) respectively for correctness in creating objects of the class since these objects have instances of the procedures, functions and variables of the parent class (Binder, 1999).The major goal of the unit test is to eliminate error at the most fundamental level of the program, thereby subjecting the code to a lot of tests is its aim, since most pieces of production level software consist of a great number of classes, which make up the modules of the program

An interface is a piece of code that exposed the Service Level Contract between classes in a program, and provides a means of interaction between them. Integration testing allows for the testing of these interfaces and allows the programmer to check for a valid specification in their creation, usage and test for errors accordingly (Xuan &Monperrus, 2014).). A major aspect of integration testing is that it is dome in progression and larger interfaces that connect modules are tested after smaller interfaces that connect classes have been tested.

In testing the entire system, otherwise known as "System testing", the software artefact as a whole is tested. This includes all user visible interfaces in the case of interactive programs or all components in the case of non-interactive programs to get an overall correct or incorrect output. In system it should be noted that the parameters for testing are of strict and utmost importance, as an improper knowledge of parameters to test will result in a false positive, which will lead to the malfunctioning of the program when other parameters are supplied (Willison, 2004).

The final level of testing is the test for "Acceptance" of the software artefact. Here the business requirements as presented by the owner are crosschecked with the technical requirements. Furthermore, the owner of the software signs off on the project as fulfilling all the use cases for the project. The product is finally installed in a production or working environment and monitored for recall incase flaws are detected in the external environment (). Most times, these flaws are due to user error and not attributable to the programmer.

Programming languages have a role in software testing. This is because modern software must conform to standards which supersede, in cases of production level software, user defined and arbitrary standards. These standards allow for cohesion among members of the software development team beyond test creation as other criteria can be introduced in the software development process to ascertain the correctness of the software as a whole. Each programming language adhered to standards, and for example a language such as C++ is closely monitored by its standards organization for compliance with code written in the language. Every project developed in the Java Programming Language also adheres to a standard as the documentation contained in the languages instructs programmers on how to write source code, interfaces, test the code and deliver production ready code that meets business requirements. These rules include a formal and well defined syntax, clear and unambiguous semantics, a means of enforcing a defined code structure through code blocks or indentation and code typing enforcing through requiring that variables be aligned to class types at the point of declaration (Ransome & Misra, 2013).Further roles programming languages have on software testing is by enforcing the design and use of the compiler as source code is invalid until it meets the specifications set out in the design.This role by programming languages has led to

their definition in proxy of international standards and the mandate that certain critical programs be developed in only an approved list of programming languages which meet internationally defined criteria of "correctness" or the assurance by the appropriate developer to the appropriate regulatory authority that production level software meets the safety and reliability metrics of the environment in which it is to be deployed.

Finally, some standards organizations such as the IEEE have published standards handbooks on code quality metrics for programmers to measure their code against, these serve as manual testing guidelines and are used by bodies like contractors with non-skilled personnel to vet the programs they purchase from developers. This makes the agile methodology even more important as the code and test mechanism means test cases which are software methods to simulate code scenarios are actually written into the business requirement use cases and do not exist only in the developer use case (Ministry of Defence, 1991).

Statement of the Problem

It is obvious that in the world of software engineering, programming languages have a role to play in proving the validity of software through tests. However, these tests are also a challenge for programming languages as they often lack the tools to automate them beyond Unit tests and so are restricted to Quality Assurance manuals such as the Ministry of Defense handbook for software quality. This report shows how the problem of testing can be addressed in a programming language using Integrated Development Environment tools as well solving the problem of automated testing, and validates the need for suitability as a criterion for assessment of a language for fitness of purpose when programming for a particular industry,

Aim and objectives

The aim of this report is to prove the need for agile methodology in software testing and also to stratify agile methods by creating a formal framework where the traditional tests such as the Unit, integration, system, acceptance, white box, and black box tests are carried out in a formal environment. To achieve this, the objectives are:

1. Use an Integrated Development Environment to develop a non0trivial piece of software.
2. Attempt to prove the software for validity using the software provided for that programming language using automated testing methods.
3. Explain the ability or otherwise for the software to be proved for validity using automated testing but rather manual testing.

Materials and Methodology

The materials include the Microsoft Visual Studio Community Integrated Development Environment 2019 edition which is a Microsoft free IDE for software development in the C sharp (C#) programming language. Microsoft is a Defense Advanced Research Projects Agency partner and as such adheres to the standards outlined in the Software Handbook for assessment of software. C# is an OOP 4^{th} generation language and as such is suitable for agile development, however it is also to be tested for its suitability for Unit, Integration, System, white and black box tests. The methodology is to create a program such as a Bank application using Visual Studio, create a test suite for unit testing the methods in the Bank application and confirm that integration tests, system tests and acceptance tests cannot be performed without integrating third party software but rather a Quality Assessment checklist is made for the remaining aspects of testing

Implementation of the Study

In step 1, you will start the Microsoft Visual Studio Integrated Development Environment.

In step 2, with the start window open, you will choose the option to begina project

In step 3 you will find the C sharp (C#) template for projects

In step 4 you will give the project a name. For out non trivial application we choose a Bank application.

In step 5 you give the project a name and a target framework by renaming the default c sharp file name to the BankAccount file name, open the code editor and type the corresponding code that created the variables and methods for the BankAccount class as seen in the diagram below:

```
using System;

namespace BankAccountNS
{
    /// <summary>
    /// Bank account demo class.
    /// </summary>
    public class BankAccount
    {
        private readonly string m_customerName;
        private double m_balance;

        private BankAccount() { }

        public BankAccount(string customerName, double balance)
        {
            m_customerName = customerName;
            m_balance = balance;
        }

        public string CustomerName
        {
            get { return m_customerName; }
        }

        public double Balance
        {
            get { return m_balance; }
        }

        public void Debit(double amount)
        {
            if (amount > m_balance)
            {
                throw new ArgumentOutOfRangeException("amount");
            }
```

Figure 1: Source Code for the BankAccount Class

In step 6 you build the project and this is done by choosing the build solution option in the Debug menu.

This process has resulted in the development of a class of code and data where we can perform tests of the code and data using white box tests since we have access to the source code.

The first test to be performed are unit tests.

To create the unit test we proceed as follows:

First the option to add a new project to the existing project is selected from the file menu

Next the option unit test and C# are chosen from the template option respectively

Next the project is given a name, the suitable name we are giving in this example is Bankts for Bank Test class

Next we choose a targeting framework as the .NET 4 framework and add a reference to the build from the BankAccount class so the Bankts class has access to the code and data from the BankAccount class.

Next the code to test the BankAccount class is typed into the BankAccountTest class and it is shown in the diagram below:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace BankTests
{
    [TestClass]
    public class BankAccountTests
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

Figure 2: Code for the BankAccountTest Class

Normally the first test you run is a failing test, this is done by deliberately entering false values in the asserts method for the methods in the BankAccount class. The asserts method works by taking three parameters, the method, the expected value and the actual value. This enables the tester validate the return output of the method for algorithmic correctness. After the failing test is performed, the passing test is performed whereby a correct set of values are passed to the assert method and the Integrated Development Environment employs the build tool to build and execute the unit tests in a white box approach as seen in the following diagram:

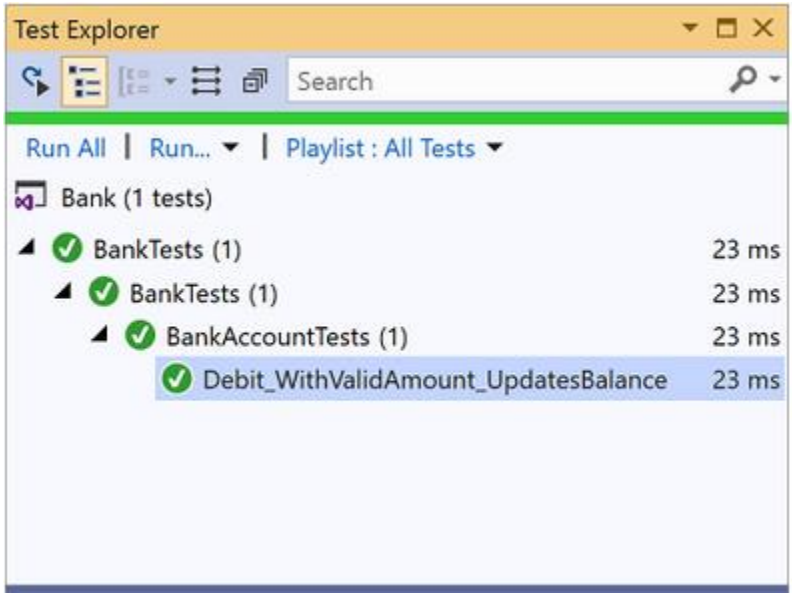

Figure 3: Passing Tests from the BankAccountTest Class

Next, the unit test is refactored to throw appropriate error methods in addition to the assert method and this is done by using conditional statements with the appropriate System Error Class methods in the C sharp (C#) programming language to make the code robust and enable more unit testing as seen in the diagram below:

```
[TestMethod]
public void Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRange()
{
    // Arrange
    double beginningBalance = 11.99;
    double debitAmount = -100.00;
    BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);

    // Act and assert
    Assert.ThrowsException<System.ArgumentOutOfRangeException>(() => account.Debit(debitAmount));
}
```

Figure 4: Refactoring for Exceptions and Conditionals

Finally, the Visual studio does not provide for tests beyond unit tests, so system, integration and acceptance tests must be done with third party frameworks which may not be written in the C# language, This defeats the purpose of our report as these frameworks may in themselves have hidden bugs and so we are restricted to white box unit testing of the code in C# when using the recommended Integrated Development Environment which is the Visual Studio.

Results of the Study

The results for the C sharp programming language using the Visual Studio Integrated Development Environment can be seen in Table 1:

Table 1: Test and Corresponding Programming Role Suitability

Test	Programming language Role
Unit test	Valid, white box test testing of methods and datatypes
Integration test	Valid, black box testing of interfaces with third party frameworks
System test	Valid black box testing of entire application with QA manuals
Acceptance test	Valid, black box testing of entire application with Business Requirements documentation
Usability test	Valid, black box testing of Look and Feel for entire visible part of the application using Software Requirements and Business Requirements
Accessibility test	Valid, black box testing using an interpretation of the Disability Act document

The unit test is transparent and is easily executed by the programming language and its associated Integrated Development Environment. The integration test is a black box test and is only executable by means of testing third party frameworks. The system test is a black box test and is executed using Quality Assurance documentation from the developer use cases and other sources. The Acceptance test is a black box test that is carried out by means of the Requirements given by the Business Owner. The Usability test is given by a translation of the Software Requirements and Business Requirements and is a black box test. Finally, the Accessibility test is a black box test which checks for compliance of the application for use with disabled individuals.

Discussion

The twelve principles of agile development are important in software development because they emphasize the importance of the harmonization of effort between the software developer and the business owner in drawing up the developer requirements and the business requirements of the document for the software project to be executed on a timely, modular, cost effective and technically sound basis (Beck et al., 2011). The use of tests for validating software is valid because they are a transparent way of assessing the effort of the development team carrying out various parts of the project, from static analysis of code to analysis of code flow, from assessment of integration of system components to analysis of the entire system including servers, databases and version control systems. These tests also include an analysis of the look and feel and the compliance of the application with disabled people and the different disabilities (Kaner, 2006).

Some programming languages in recent years have built in test frameworks but are weakly typed and since they tend as a result to be procedural in method flow, have the deficiency of poor code flow and an increased use of black box testing methods of boundary value estimation, true false value pairs testing, static flow observation and the use of truth tables and classic programming test methods for manual testing in preference to their automated methods which are not independently verifiable. As a result, there is need to take into consideration the tradeoff between strongly typed and weakly typed languages in test first development in agile software development. Programming languages are developed to international standards such as the Defense Research Projects Agency standard which emphasizes the use of strongly typed languages, and as a result, each programming language adheres to its individual standard with respect to typing syntax and interfaces which in the case of strongly typed languages is available only for unit tests. It therefore is an aberration to test code written in one programming language with tools developed in another programming language as one may introduce unknown bugs into the application by so doing ((Ransome&Misra, 2013). These tools may in themselves have flaws as they may not have automated testing tools in themselves for integration and system testing but are only applications for running other applications and storing output in server and log files.

Conclusion

In this report, the agile programming methodology has been presented. The need for the harmonization pf effort between the programmer developer requirements and the business owner requirements has been explained. The need for various tests in the execution of a complete software project has been explained. The role of programming language features such as object orientation, static typing and code flow in testing has been explained, the implementation of a test driven software application to demonstrate the importance of white box testing has also been undertaken. The results of the different types of tests and their importance has been explained, and the tradeoff between strong typing and weakly typed languages, the emphasis of white box testing and the relegation of third party unit frameworks in favor of classic black box testing has been analyzed.

References

1. Beck, K., et al. (2001). The Agile Manifesto. Agile Alliance. http://agilemanifesto.org/
2. Binder, Robert V. (1999). *Testing Object-Oriented Systems: Objects, Patterns, and Tools* (https://archive.org/details/testingobjectori00bind/page/45). Addison-Wesley Professional. p. 45 (https://archive.org/details/testingobjectori00bind/page/45).
3. Black, R. (2011). *Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional* (https://books.google.com/books?id=n-bTHNW97kYC&pg=PA44). John Wiley & Sons. pp. 44–6.
4. Cambridge Online Dictionary (n.d). Testing. https://dictionary.cambridge.org/dictionary/english/testing
5. Graham, D.; Van Veenendaal, E.; Evans, I. (2008). *Foundations of Software Testing* (https://books.google.com/books?id=Ss62LSqCa1MC&pg=PA57). Cengage Learning. pp. 57–58.
6. Kaner, C. (2006). *Exploratory Testing* (http://www.kaner.com/pdfs/ETatQAI.pdf) (PDF). Quality Assurance Institute Worldwide Annual Software Testing Conference. Orlando, FL.
7. Kaner, C.; Falk, J., & Nguyen, H. Q. (1999). *Testing Computer Software* (2nd ed.). New York: John Wiley and Sons. ISBN 978-0-471-35846-6.
8. Limaye, M.G. (2009). *Software Testing* (https://books.google.com/books?id=zUm8My7SiakC&pg=PA108). Tata McGraw-Hill Education. pp. 108–11
9. Ministry of Defence (1991). ``The Procurement of Safety Critical Software in Defence Equipment", (Part1: Requirements; Part2: Guidance). Interim Defence Standard 00-55.
10. Ransome, J. &Misra, A. (2013). *Core Software Security: Security at the Source* (https://books.google.com/books?id=MX5cAgAAQBAJ&pg=PA140). CRC Press. pp. 140–3.
11. Oberkampf, W.L.; Roy, C.J. (2010). *Verification and Validation in Scientific Computing* (https://books.google.com/books?id=7d26zLEJ1FUC&pg=PA155). Cambridge University Press. pp. 154–5.

12. Willison, J. S. (2004). "Agile Software Development for an Agile Force" (https://web.archive.org/web/20051029135922/http://www.stsc.hill.af.mil/crosstalk/2004/04/0404willison.html).
13. Woods, Anthony J. (June 5, 2015). "Operational Acceptance – an application of the ISO 29119 Software Testing standard" (https://www.scribd.com/document/257086897/Operational-Acceptance-Test-White-Paper-2015-Capgemini)
14. Xuan, J.; & Monperrus, M. (2014). "Test case purification for improving fault localization". *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of SoftwareEngineering - FSE 2014*: 52–63. arXiv:1409.3176 (https://arxiv.org/abs/1409.3176).