American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-6, Issue-1, pp-271-275
www.ajer.org
Research Paper Open Access

Software Testing Algorithm Units

Ashiqur Rahman[1],Ferdaush Hasan Sunny[2],Hasan Mahmud Mishu[3], Farjana Sumi[4]
[1](M.Sc in Information Technology (IT), Jahangirnagar University, Bangladesh)
[2,3, 4](MCSE, Royal University of Dhaka, Bangladesh)

ABSTRACT: *Software testing is a process used to identify the correctness, completeness and quality of developed computer software. Software Testing is important as it may cause mission failure, impact on operational performance and reliability if not done properly. Effective software testing delivers quality software products satisfying user's requirements, needs and expectations. Software testing in the field of Software Engineering is a process in the life-cycle of a software project that verifies that the product or service meets quality expectations and validates that software meets the requirements specification. Software testing is intended to locate defects in a program, although a given testing method cannot guarantee to locate all defects. As such, it is common for an application to be subjected to a range of testing methodologies throughout the software life-cycle, such as unit testing during development, integration testing once modules and systems are completed, and user acceptance testing to allow the stakeholders to determine if their needs have been met.It is the process of executing a program / application under positive and negative conditions by manual or automated means. It checks for the:-*

- *Specification*
- *Functionality*
- *Performance*

Keywords: *Junits, Rubby, Frameworks, Black boxes, code coverage.*

I. INTRODUCTION

Unit testing is a type of software testing that involves the preparation of well-defined procedural tests of discrete functionality of a program that provide confidence that a module or function behaves as intended. Unit tests are referred to as 'white-box' tests (contrasted to 'black-box' tests) because they are written with full knowledge of the internal structure of the functions and modules under tests. Unit tests are typically prepared by the developer that wrote the code under test and are commonly automated, themselves written as small programmers that are executed by a unit testing framework (such as JUnit for Java or the Test framework in Ruby). The objective is not to test each path of execution within a unit (called complete-test or complete-code coverage), but instead to focus tests on areas of risk, uncertainty, or criticality. Each test focuses on one aspect of the code (test one thing) and are commonly organized into test suites of commonality.
Some of the benefits of unit testing include:

- **Documentation:** The preparation of a suite of tests for a given system provide a type of programming documentation highlighting the expected behavior of functions and modules and providing examples of how to interact with key components.
- **Readability:** Unit testing encourages a programming style of small modules, clear input and output and fewer inter-component dependencies. Code written for easy of testing (testability) may be easier to read and follow.
- **Regression:** Together, the suite of tests can be executed as a regression-test of the system. The automation of the tests means that any defects caused by changes to the code can easily be identified. When a defect is found that slipped through, a new test can be written to ensure it will be identified in the future.

Unit tests were traditionally written after the program was completed. A popular alternative is to prepare the tests before the functionality of the application is prepared, called Test-First or Test-Driven Development (TDD). In this method, the tests are written and executed, failing until the application functionality is written to make the test pass. The early preparation of tests allow the programmer to consider the behavior required from the program and the interfaces and functions the program needs to expose before they are written.

The concerns of software testing are very relevant to the development, investigation, and application of Metaheuristic and Computational Intelligence algorithms. In particular, the strong culture of empirical investigation and prototype-based development demands a baseline level of trust in the systems that are presented in articles and papers. Trust can be instilled in an algorithm by assessing the quality of the algorithm implementation itself. Unit testing is lightweight (requiring only the writing of automated test code) and meets the needs of promoting quality and trust in the code while prototyping and developing algorithms. It is strongly suggested as a step in the process of empirical algorithm research in the fields of Metaheuristics, Computational Intelligence, and Biologically Inspired Computation.

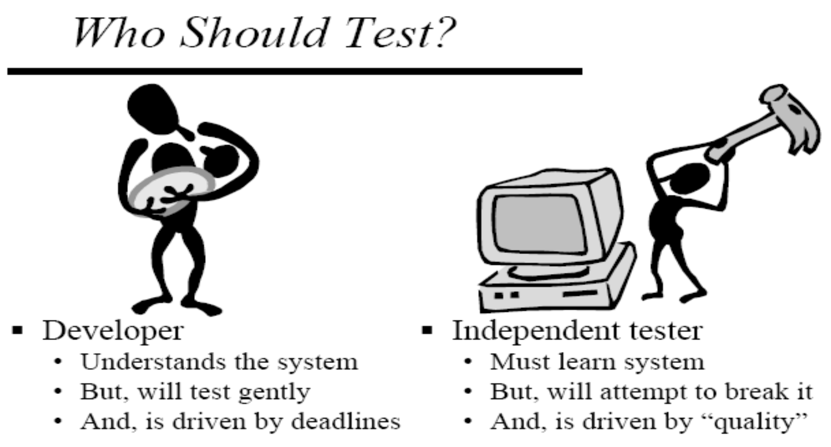
II. TESTING TEAM

✓ **Program Manager-**
- The planning and execution of the project to ensure the successof a project minimizing risk throughout the lifetime of the project.
- Responsible for writing the product specification, managing the schedule and making the critical decisions and trade-offs.

✓ **QA Lead-**
- Coach and mentor other team members to help improve QA effectiveness
- Work with other department representatives to collaborate on joint projects and initiatives
- Implement industry best practices related to testing automation and to streamline the QA Department.

✓ **Test Analyst\Lead-**
- Responsible for planning, developing and executing automated test systems, manual test plans and regressions test plans.
- Identifying the Target Test Items to be evaluated by the test effort
- Defining the appropriate tests required and any associated Test Data
- Gathering and managing the Test Data
- Evaluating the outcome of each test cycle

✓ **Test Engineer-**
- Writing and executing test cases and Reporting defects
- Test engineers are also responsible for determining the best way a test can be performed in order to achieve 100% test coverage of all components



Who Should Test?

- Developer
  - Understands the system
  - But, will test gently
  - And, is driven by deadlines
- Independent tester
  - Must learn system
  - But, will attempt to break it
  - And, is driven by "quality"

III. TESTING CYCLE

A test plan is a systematic approach to testing a system i.e. software. The plan typically contains a detailed understanding of what the eventual testing workflow will be.

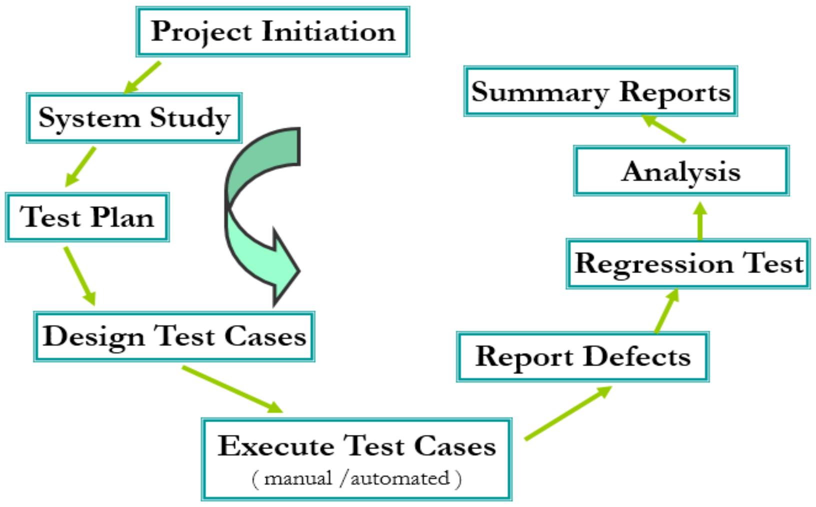


Fig 1: Testing Cycle

Listing (below) in provides the source code for the Genetic Algorithm in the Ruby Programming Language. Important considerations when in using the Ruby test framework, is ensuring that the functions of the algorithm are exposed for testing and that the algorithm demonstration itself does not execute. This is achieved through the use of the (if __FILE__ == $0) condition, which ensures the example only executes when the file is called directly, allowing the functions to be imported and executed independently by a unit test script. The algorithm is very modular with its behavior partitioned into small functions, most of which are independently testable. The reproduce function has some dependencies although its orchestration of sub-functions is still testable. The search function is the only monolithic function, which both depends on all other functions in the implementation (directly or indirectly) and hence is difficult to unit test. At best, the search function may be a case for system testing addressing functional requirements, such as "does the algorithm deliver optimized solutions".

IV. TESTING SYSTEM

Smoke Testing:

Smoke testing is non-exhaustive software testing, ascertaining that the most crucial functions of a program work, but not bothering with finer details.

Alpha Testing:

1. The application is tested by the users who doesn’t know about the application.
2. Done at developer’s site under controlled conditions
3. Under the supervision of the developers.

Acceptance Testing:

A formal test conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.It is the final test action before deploying the software. The goal of acceptance testing is to verify that the software is ready and can be used by the end user to perform the functions for which the software was built.

Beta Testing:

1. This Testing is done before the final release of the software to end-users.
2. Before the final release of the software is released to users for testing where there will be no controlled conditions and the user here is free enough to do whatever he wants to do on the system to find errors.

Regression Testing:

Testing with the intent of determining if bug fixes have been successful and have not created any new problems. Also, this type of testing is done to ensure that no degradation of baseline functionality has occurred.

Money Testing:

Testing the application randomly like hitting keys irregularly and try to breakdown the system there is no specific test cases and scenarios for monkey testing.

V. QA VS. QC

• Quality Assurance makes sure that we are doing the right things, the right Way.	• Quality Control makes sure the results of what we've done are what we expected .
• QA focuses on building in quality and hence preventing defects.	• QC focuses on testing for quality and hence detecting defects.
• QA deals with process.	• QC deals with product.
• QA is for entire life cycle.	• QC is for testing part in SDLC.
• QA is preventive process.	• QC is corrective process.

VI. BUG LIFE CYCLE

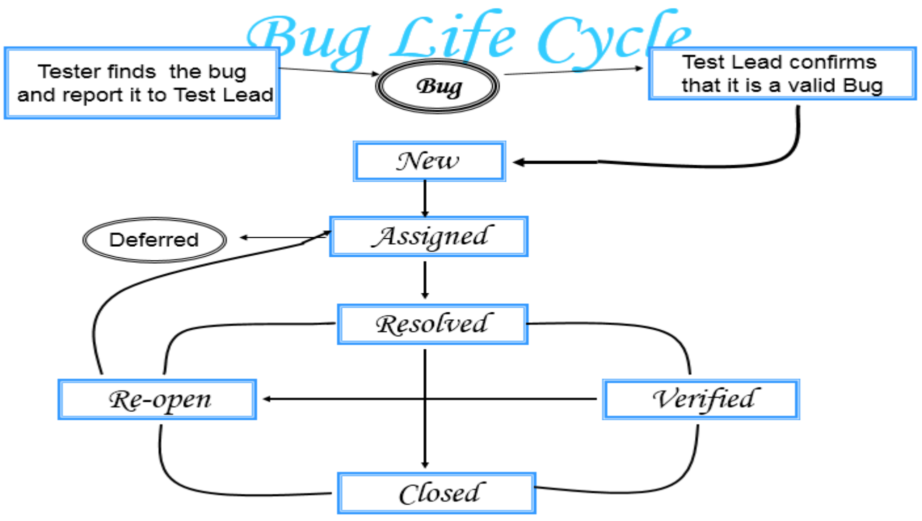

Bug Life Cycle
Tester finds the bug and report it to Test Lead
Bug
Test Lead confirms that it is a valid Bug
New
Deferred
Assigned
Resolved
Re-open
Verified
Closed

Fig 2: BUG life cycle

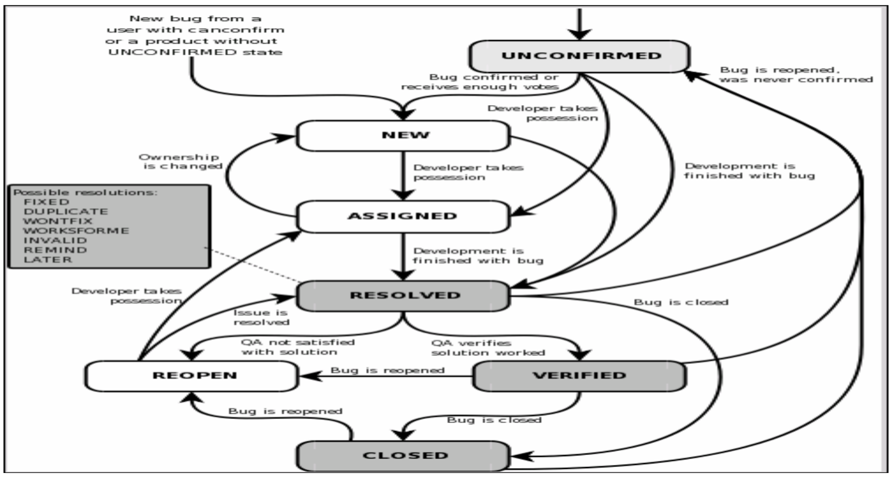

New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug confirmed or receives enough votes
Bug is reopened, was never confirmed
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions: FIXED DUPLICATE WONTFIX WORKSFORME INVALID REMIND LATER
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Bug is closed
Issue is resolved
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

Fig 3: Operation Process

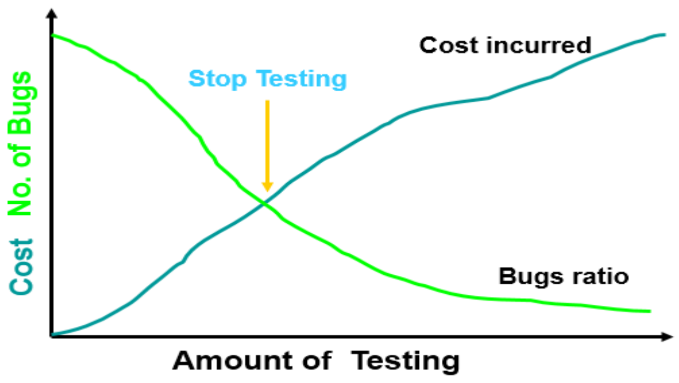


Fig 4: Testing VS Bugs

VII. CONCLUSION

The tests for probabilistic expectations is a weaker form of unit testing that can be used to either provide additional confidence to deterministically tested functions, or to be used as a last resort when direct methods cannot be used. Given that a unit test should 'test one thing' it is common for a given function to have more than one unit tests. The reproduce function is a good example of this with three tests in the suite. This is because it is a larger function with behavior called in dependent functions which is varied based on parameters. Listing (below) provides the TC_GeneticAlgorithm class that makes use of the built-in Ruby unit testing framework by extending the TestCase class. The listing provides an example of ten unit tests for six of the functions in the Genetic Algorithm implementation. Two types of unit tests are provided:

- **Deterministic:** Directly test the function in question, addressing questions such as: does onemax add correctly? and does point_mutation behave correctly?
- **Probabilistic:** Test the probabilistic properties of the function in question, addressing questions such as: does random_bitstring provide an expected 50/50 mixture of 1s and 0s over a large number of cases? and does point_mutation make an expected number of changes over a large number of cases?

REFERENCES

[1]. Yogesh Singh “Software Testing” Cambridge University Press ISBN 978-1-10701269-7 First edition 2012.
[2]. Dr.S.S.Riaz Ahamed “Studying The Feasibility and Importance of Software Testing: An Analysis” International Journal of Engineering Science and Technology Vol.1(3), 2009, ISSN: 0975-5462 pp.119-128.
[3]. Prof. (Dr.) V. N. Maurya, Er. Rajender Kumar “ Analytical Study on Manual vs. Automated Testing Using with Simplistic Cost Model” International Journal of Electronics and Electrical Engineering ISSN : 2277-7040 Volume 2 Issue 1 (January 2012)pp.24-35.
[4]. Prof. Sankar “Mc Graw Hill Education” QuickTest Professional.
[5]. Fewster, M., Graham, D., and Software Test Automation: Effective Use of Text ExecutionTool, AddisonWesley, 1999.