The International Journal of Engineering and Science (IJES)
|| Volume || 9 || Issue || 09 || Series I || Pages || PP 52-57 || 2020 ||
ISSN (e): 2319-1813 ISSN (p): 20-24-1805

# Designing Test Driven Development Architecture for Unit Testing of Object-Oriented Software

[1]Mohammed SDhallaComputer , [2]Abdulrahman Issa Kh Shybub1
*[12] DepartmentHigher Institute of Science and Technologie In Nalut*

**--------------------------------------------------------- *ABSTRACT*---------------------------------------------------------**
*This paper focuses on addressing theproblems and challenges involved in testing and Improving the quality of object-oriented test case design. The aim of the study is to provide the software tester/developer with quantitative reliability measures of the automate test case design for unit testing of object-oriented system. The paper introduces automated test case design architecture for unit testing of OOS and describes the problems and challenges involved in unit testing of object-oriented system.*

***KEYWORDS-****OOS, TDD, XP, BI{(ck~Box Testing, IW, VI, Test Case*

Date of Submission: 30-09-2020 Date of Acceptance: 13-10-2020

## I. INTRODUCTION

"Testing proves the presence, not the absence of hugs."E.W. Dijkstra "Absence of evidence is not evidence of absence."- Source unknownTesting is an essential part of software development providing an indicator of the quality ofthe software [ 1 ]. \
The Unit Test is the lowest level of testing performed during software development, where individual units of software are tested in isolation from other parts of program/software system.
Automated Testing is a program which be a job stream, that runs the program being tested, feeding it the proper input, and checking the output against the output that was expected. Once the test case is written, 110 human interaction is needed either to run the program or to look to see if it worked, the test case doses all that, and somehowindicate[4).
Test Script/Driver is a set of conditions specified Oil each input variable for guiding the automatic test rase design.
A test Suite or a Test Run is a set of test cases that executed sequentially.

## II. TESTING METHODOLOGIES

Organizations are still asking how they can predict the quality of their software before it is used despite the substantial research effort spent attempting to find an answer to this question over the last 30 years. There are many papers advocating statistical models and metrics which purport to answer the quality question. Defects, like quality, can be defined in many different ways bu; are more commonly defined as deviations from specifications or expectations which might lead to failures in operation.
Generally, efforts have tended to concentrate on the following three problem perspectives:

1- Predicting the number of defects in the system;
2- Estimating the reliability of the system in terms often to failure;
3- Understanding the impact of design and testing processes on defect counts and failure densities.

### A.Develop unit tests in parallel or before implementation

The Parallel Architecture for Component Testing (PACT) provides this organization. The (PACT) architecture shown ill fig, is highly adaptable, providing various level coverage, applicable to various platforms and languages, and maintains a distinction between production code and testing code. The separation of production code from test code ismaintained by creating classes that are separate from the production classes, but that hold the test cases for corresponding classes in the production architecture.

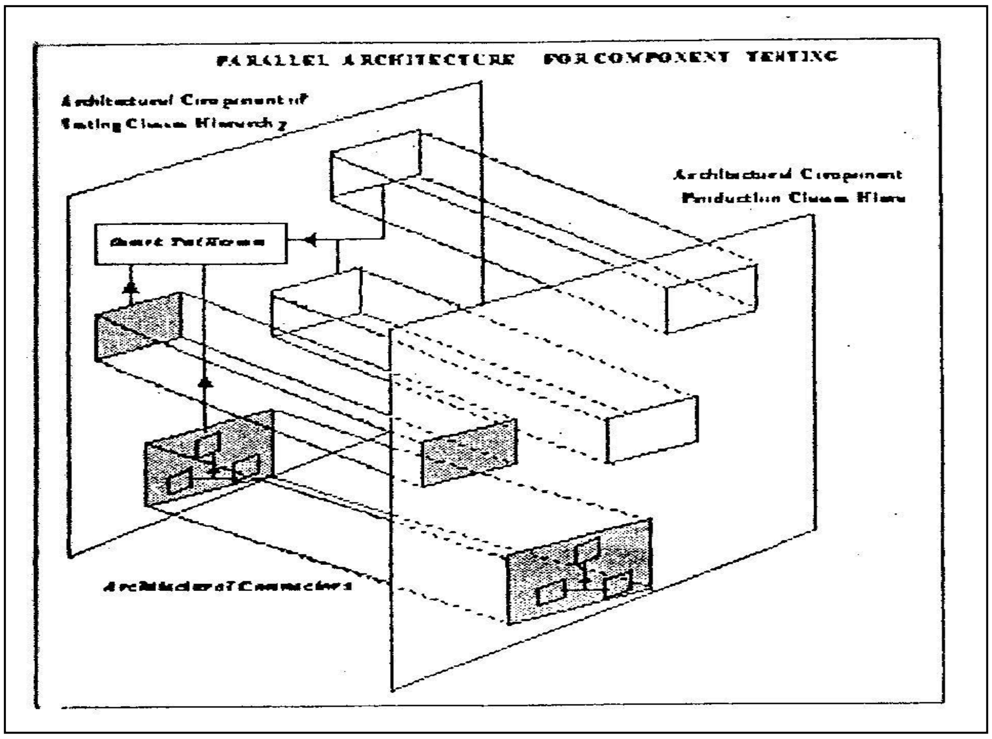
PARALLEL ARCHITECTURE FOR COMPONENT TESTING
Architectural Component of Testing Class Hierarchy
Architectural Component Production Class Hier
Architectural Connectors

**Figure I.** Parallel Architecture for Component Testing

***B. ExtremeProgramming***

Popularized by Extreme Programming (XP), the concept of developing unit tests prior to the actual Koftware itself is a useful one. Using this approach, it is necessary to have requirement defined prior to the development of unit tests, since they will be used as Ihc guide for unit test development [3].

There are many benefits to developing unit test prior to actuallymplemenation a software component.

1. First and most obvious, is that the software will be considered complete when it provides the functionality required to successfully execute the unit test, and no less.

In this way, the software is developed to meet the requirement, and that requirement isstrictly enforced lindchecked by the unit test.

2.The second benefit is the effect of focusing the developer's efforts on satisfying the exact problem, rather than developing a larger solution that also happens to satisfy the requirement.

This allows the smallest possible solution to be developed, and will most likely result in less code and It more straightforward implementation.

3.The third one more subtle benefit, is that is there is any question as to the developer'S interpretation of the requirements, it will reflected in (he unit test code.

This provide a useful reference point . for determining what the code was intended to do by the developer, versus what it is supported to do according to the requirement.

To efficiently take advantage of this technique. the requirement documentation must be present and for the most part complete prior to development. This is usually regarded as the best approach, since developing prior to the completion of requirements for a particular function can be risky. Requirements should be specified at a somewhat detailed level. allowing for the required objects and functions to be easily deterrriirv-'. From the requirement documentation, the developer can layout a general unit test strategy for the component, including success and failure tests.

***C. Test Driven Development***

Test Driven Development (TDD) is a new approach that offers the potential to significantly improve the state or software construction. TDD is a discipline software development practice that focuses on software design by writing automated unit tests followed by production code in short, frequent iteration [2]. TDD focuses the developer's attention on a software's interface and behavior while growing the software architecture organically.

TDD has gained recent attention with the popularity of the Extreme Programming[2]agile software

development methodology.
The empirical studies to date have focused on TDD as a testing technique lor comprehensive evaluation of how TDO affect overall software architecture qual ity beyond just defect density.

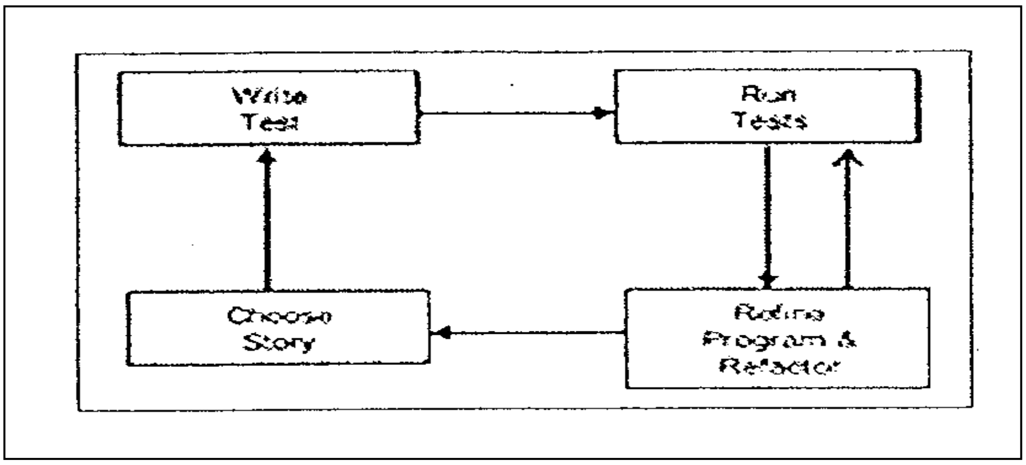

**Figure 2**. Model of TDD

Test Driven Development involves the evolving design of the whole system within an organisational context, with iterations and the planning game:[7]
•A customer writes stories that will lead to the development of the system.
The developers estimate the cost of developing the stories (which may require that a story is broken down into smaller. more manageable stories ).
•The estimates help the customer to prioritise the stories. in order to choose which will fit into the next iteration.
•Customer tests are developed for the stories chosen for iteration. They are used to determine whether a story is complete.
•Stories may be exploratory because no-one is yet clear as to what the system should do, or they may be well understood by the customer and clear}, part of the system. As the system evolves. the value of different possible aspects of the system will become clearer as it begins to be fitted into its organisational niche.

## III. UNIT TEST ARCHITECTURE

Proper testing of an application requires more than simply verifying the simulated or re-created user actions. Testing the unit through the user interface only. without understanding component's internal structure, is typically referred to as black-box testing. By itself, black-box testing is not the most effective way to test. In order to design and implement the most effective strategy for thoroughly investigating the correct functioning of an unit, the tester/developer team must have a certain degree of knowledge of the component's internals, such as its major architectural components. Such knowledge enables the testing team to design better unit tests and perform more effective defect diagnosis. Testing a unit or application by directly targeting the various layers of the system is referred to as gray-box testing.

In a large system the TDD approach. the code is usually developed in a class layer fashion. by dividing functionality into several layers, each of which is responsible for a certain aspect of the system. For example, a system could make usc of the following layers in its implementation.

**Database abstraction:**
An abstraction for database operations. this layer wraps up database interaction into a set of classes or components that are called by code in other layers to interact with the database.

**Domain Objects**
A set of classes that represent entities in the system's problem domain. such as "Account" or an "Order". Domain objects typically interact with the database layer. A domain object will contain a small amount of code logic. and may be represented by one or more database tables.

**Business Processing**
Components, or classes, that implement business functionality that makes use of one or more domain objects to accomplish a business goal (transactions), such as "Place order" or "Create Customer Account".

**User Interface**

The user visible components of the application that are use to interact with the system. This class can be implemented in a variety of ways, but may manifest itself as a window with a simple command line interface. This class is typically at the "top" of the system's partitions.

The above list is a somewhat simplified example of a layered implementation, but it demonstrates the separation of functionality across layer from a "bottom-up" perspective. Each layer will be made up of several code modules that is called class partitions, which work together to perform the functionality of the layer. During the development/Testing of such a system, it is usually most effective to assign testers/developers to work with a single layer, and communicate with testers/developers, and components, in other layers through a documented and defined interface. So, in a typical interaction, the user chooses to perform some action in the user interface, and the user interface layer calls the Business

Processing (BP) layer to carry out the action. Internally, the BP layer uses domain objects and other logic to process the request on behalf of the user interface (UI) layer. During the course of this processing, the domain objects will interact with the database abstraction layer retrieve or update information in the database. There are many advantages to this approach. including the separation of testers/developers across the layers, a defined interface for performing work, and the increased potential for reusing layers and classes (code).

## IV. DESIGN OF MOOEL

Implementation of the design is a straightforward matter of translating the design into code. since most difficult decisions are made during design. The code should be a simple translation of the design decision into the peculiarities of a peculiar language. Decisions do have to be made while writing code. but each one should affecton)y a small part of the program so they can be changed easily.

Test Driven Development involved analysis ,design implementation and testing are very interleaved activities performed in an incremental fashion .

Here. I show figures of a unit and of an aggregation of units. which I will call a subsystem[5].

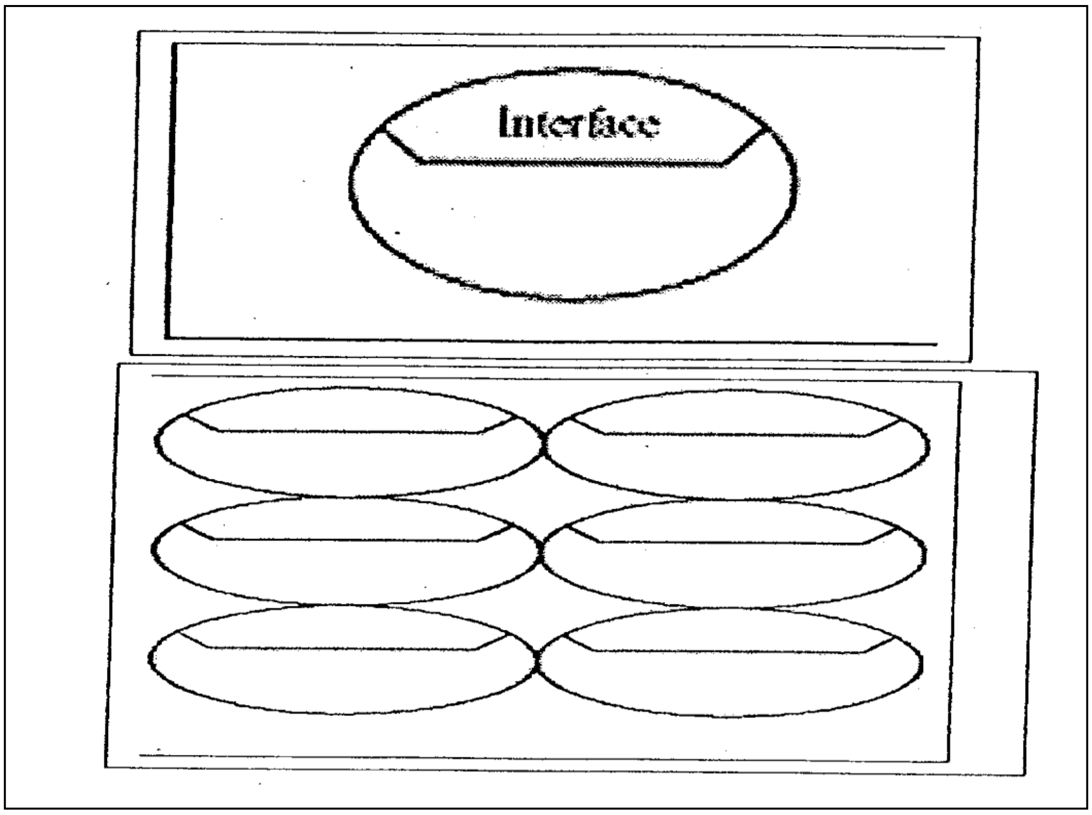

**Figure 3.** A unit and Aggregation of Units

As we seen the V model says That someone should first test each unit. When all the subsystem's units are tested, they should be aggregated the subsystem tested to see if it works as a whole.

So how do we test the unit? We look at its interface as specified in the detailed design, or at the code, or at both, pick inputs that satisfy some test design criteria, feed those inputs to the interface, then check the results for correctness. Because the unit usually can not be executed in isolation, we have to surround it with stubs and drivers, as in the fig. The arrow represents the execution trace of a test.

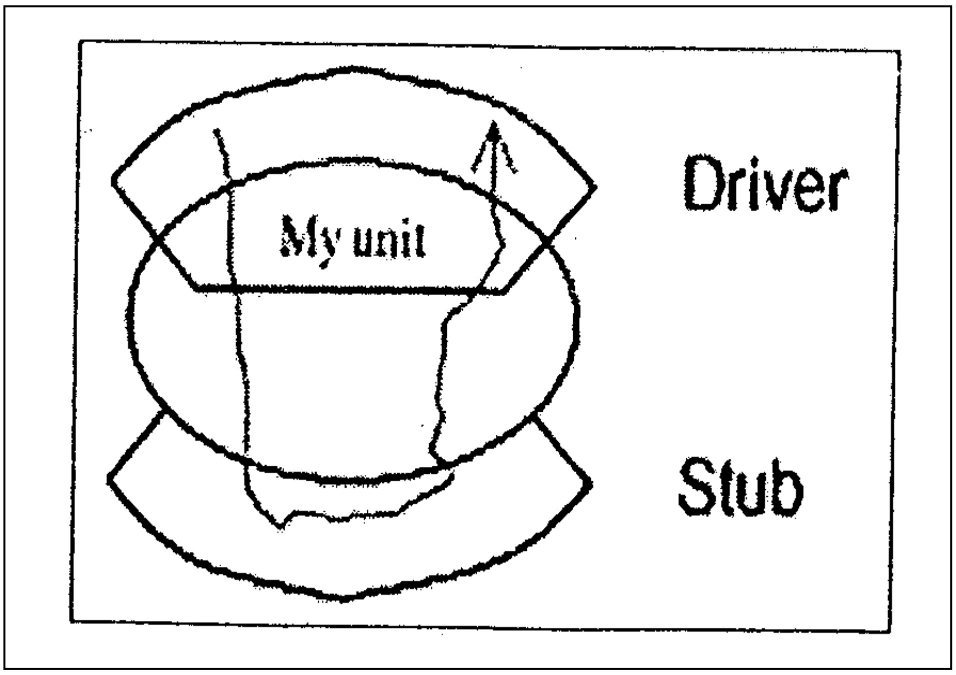

**Figure 4.** Testing of a particular unit

A test case designed to find bugs in a particular unit might be best run with the unit in isolation, surrounded by unit-specific stubs and drivers.

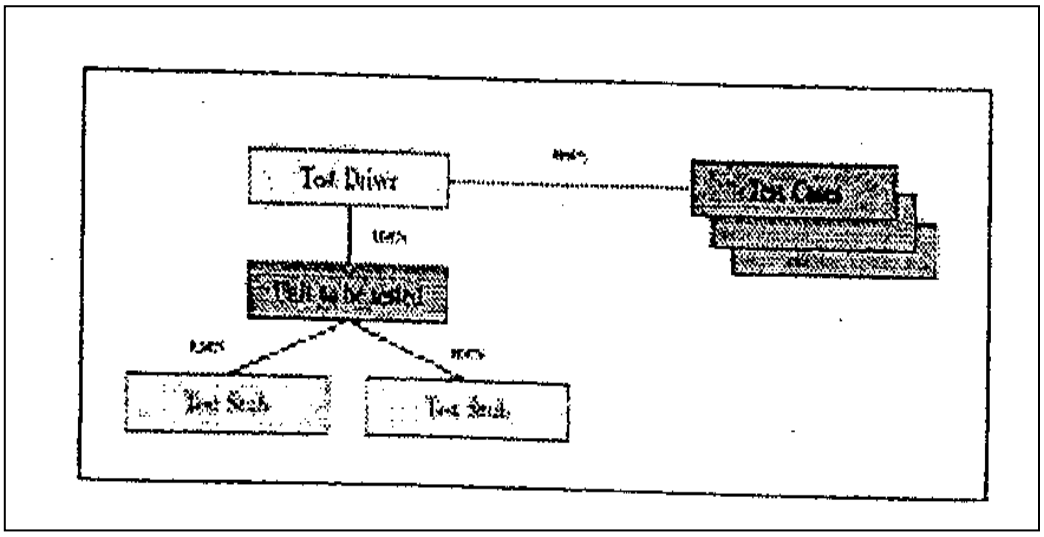

**Figure 5.** Test Case

## V. CONCLUSION

There are many benefits to developing unit tests prior to implementing a software component. The first and most obvious, is that unit testing forces development of the software to be pursued in a way that meets each requirement. The software is considered complete when it provides the functionality required to successfully execute the unit test, and not before; and the requirement is strictly enforced and checked by the unit test. A second benefit is that it focuses the developer's efforts on satisfying the exact problem, rather than developing a larger solution that also happens to satisfy the requirement. This generally results in less code and a more straightforward implementation. A third, more subtle benefit is that the unit test provides a useful reference for determining what the developer intended to accomplish (versus what the requirements state). If there is any question as to the developer's interpretation of the requirements, it will be reflected in the unit test code.

## REFRENCES

[1]. John A. Fodeh, and Niels B. Svendsen, "Release Metrics : When to Stop Testing with a clear conscience", Journal of Software Testing Professionals, March 2002.
[2]. Robert C. Martin:" Agile Software Development": Principles, Patterns and Practices, 2002.
[3]. Geor~e W., "Unit Testing : Test before you code' , Developer IQ, March 2005.
[4]. Eugene Volokh, "Automated Testin& ':': When and How", VESOFT (1990), Interact Magazine.
[5]. "New Model for test Development", Brain Marick, testing Foundation, www.testing.com.
[6]. D.Peled. Software Reliability Methods. Springer, 2001.
[7]. Rick Mugride "Test Driven Development and the Scientific Method" , University of Auckland, New Zealand

Mohammed SDhallaComputer, et. al. "Designing Test Driven Development Architecture for Unit Testing of Object-Oriented Software" The International Journal of Engineering and Science (IJES), 9(9), (2020): pp. 52-57.