Implementation of Mocks in UT Framework Development

[1]Rahul Kundu, [2]Ms. P Devaki
Student of Computer Network Engineering, Associate Professor
Information Science and Engineering,
The National Institute of Engineering, Mysuru, India

***Abstract*—Developers consider Unit testing as a basic and common practice where they can develop test cases along with regular source code. In modern Software development era testing is one of the essential part i terms of the quality of the system along with lo maintenance cost. During the development phase of th unit tests, we often have deal with major problems. On among them is the replacement of existing dependencie which forces to replace or modify the source code itsel Generally, we add an extra constructor function in th source code or use the concept of template specializatio to meet the needs. In C++, Unit test codes are generall bind together with the unit of source code we want t test. The goal is to explain the different types of techniques that prevent to minimal modification of the source code while writing the tests and how to develop a good unit test framework to maintain the quality of code with low cost and maintenance. To avoid the dependencies, we make use of Mocking technique in which we hide the basic code implementation and mocks the dependent function. This helps in easy testing and checks only the small unit of source code functionality we need to test as a single unit. Thus, verifying and validating all the possible combinations of the source code that is needed to be tested. Thus, It is important to be sure that one feature is tested at a time and should be notified if any error occurs at that point of time only. A technique of Mock objects and test Helpers is proposed that replaces source code with fake code implementations that simulate the real code.**

***Index Terms*— Mock Objects, Unit Testing, C++, Test Helpers, Low Cost and Maintenance, Constructor functions, Templates, Dependencies.**

I. INTRODUCTION

Nowadays, the demand for higher quality software is significantly increasing. Testing is one of the most commonly used techniques to improve the quality of software. During different phases of a software development process testing can be conducted at multiple levels. This paper focuses on unit testing, where the goal is to test a well-defined, isolated module commonly called as a unit.

Testing is an important part in modern techniques software development basically to enhance the system quality and decrease maintenance cost. During a UT (unit test) development we only validate the behavior of that unit under test. But, there are numerous problems that arise during the development of these tests. For example, an object is dependent on some other object which represents the internal state or there is some huge Database required or a need of a server connection. These things are highly impractical to do for testing a small unit code.

Unit test development comes under the part of Software Testing and development. Testing is an essential phase before the deployment of the application. It is suggested that earlier do the testing to get less bugs and create the error free application. Testing can of many types such as Unit testing, Integration Testing, Functional Testing, System Testing, Acceptance Testing, Regression Testing, etc. Thus, proper testing ensures that we know in advance that whatever we are delivering it is as far as part of the client's requirement and without errors.

Moreover, in C++, source code modification for testing could result in performance degradation, e.g. introducing a new runtime interface and virtual functions just because of testing might worsen the performance of the production code. Also, in legacy code bases often there are no unit tests. Refactoring such legacy code to provide tests is almost impossible because we cannot verify. Correctness without having unit tests; hence it is a vicious circle.

Our experience is that developing unit tests with Mock Objects leads to stronger tests and to better structure of both domain and test code. Unit tests written with Mock Objects have a regular format that gives the development team a common vocabulary. We believe that code should be written to make it easy to test and have found that Mock Objects is a good technique to achieve this. We have also found that refactoring Mock Objects drives down the cost of writing stub code.

Unit testing is a fundamental practice in Extreme Programming, but most non-trivial code is difficult to test in isolation. It is hard to avoid writing test suites that are complex, incomplete, and difficult to maintain and interpret. Using Mock Objects for unit testing improves both domain code and test suites. They allow unit tests to be written for everything, simplify test structure, and avoid polluting domain code with testing infrastructure.

In this paper, it is first described how Mock Objects are used for unit testing. Then, the benefits and costs of Mock Objects when writing unit tests and code. Finally, a brief pattern for using Mock Objects.

II. LITRATURE REVIEW

Unit tests Generations are an essential part of software development, and to support developers with their creation various techniques have been proposed to automatically

generate tests. To reduce the number of tests and to increase the code coverage achieved, techniques based on search-based software testing (SBST), cast test generation as an optimization problem, which then can be addressed with techniques like Genetic Algorithms (GAs).

Functional Mocking:

Mocking is a common approach in unit testing to isolate a class from its dependencies by using a replacement of a dependency class instead of the original one.

```
Class A
{
Public:
   bool AnInterface()
   {
      bool value = false;
      // Some Code…
      return value;
   }
};

Class Exec
{
Public:
   void display_Result()
   {
      A obj;
      if ( obj.AnInterface() == true)
      {
         // Some Code…
         // Display the Results
      }
};
```

Fig. 1. Source code example that cannot be tested without the use of mock objects

In the above example, if the test of the display_Result() is to be written, then the only possible way is that the function in Class A should return true value only. In the worst case, if that function never returns true then the function display_Result() in class Exec cannot be tested with respect to all its possible check values. In these cases, we move with Mocking the functions and make sure that it returns the check values needed to simulate the test. Basically. While mocking the function we make sure that it returns the all possible values and our function display_Result() can be tested from all the possibilities. Thus, we are required to only test the function display_Result() from Exec not he function from class A. Thus, test only the function which is required to be tested and mock those which comes under dependency.

Objects in the context of OOP are represented by classes in C++. Since C++ is not a strict object oriented language, we must investigate other language constructs like free functions and function templates from the viewpoint of dependency replacement. Generally speaking, a dependant C++ entity (class, function, class template or function template) can have diff erent kinds of dependencies. For instance, it may have a dependency on

- a global object (e.g. via a singleton).
- a global function (via a function call),
- an object via a pointer or reference,
- a type (e.g. via a type template parameter, or the type of a member).

Problems:

Generally, the following problems may arise when we replace dependency objects:

- Either we deprive the unit under test from the ownership of the dependency or we use a superfluous getter function.
- We add an otherwise unnecessary constructor or setter function.
- We introduce superfluous pointer semantics via a reference or smart pointer, which is harmful to cache locality, hence it reduces overall performance.
- We have to introduce an interface just for testing. This interface has virtual functions. Calling them requires extra pointer indirections and this might result in cache misses and it loses the possibility of inlining, thus it harms the overall performance.

C++ Seams:

A seam is an abstract concept introduced by Feathers as an instrument via we can alter behaviour without changing the original unit. Dependency replacement is done via seams in C++. Actually, there are four diff erent kinds of seams in C++ :

1. Link seam: Change the definition of a function via some linker specific setup.
2. Preprocessor seam: With the help of the preprocessor, redefine function names to use an alternative implementation.
3. Object seam: Based on inheritance to inject a subclass with an alternative implementation.
4. Compile seam: Inject dependencies at compile-time through template parameters.

III. EXISTING SYSTEM

In this section, the techniques of dependency replacement are mentioned and principles of dependency replacement in object-oriented programming and the existing techniques of dependency replacement in C++.

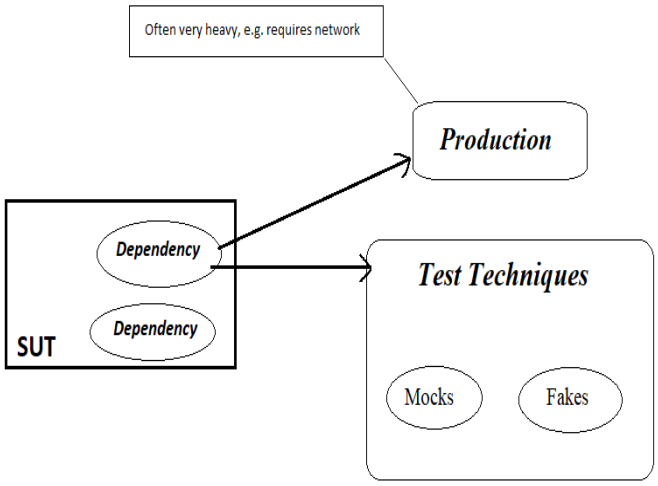

Fig. 2. Existing Techniques

In the above diagram, Fig. 2 shows the typical object under test, its dependencies and their possible replacements. If A and B are objects and "A depends on B", then we say that A is a dependent of B and B is a dependency of A. As for dependency replacement the dependent object is referred as the system under test (SUT). Sometimes we refer to that as the unit under test. **Fake** classes provide empty definitions of functions in a way that the unit tests can pass. Fakes are the simplest doubles to cut down dependencies. In object-oriented programming, **mock objects** are simulated objects that mimic the behavior of real objects in controlled ways. A programmer typically creates a mock object to test the behavior of some other object, in much the same way that a car designer uses a crash test dummy to simulate the dynamic behavior of a human in vehicle impacts.

IV. Proposed Approach

An important aspect of Extreme Programming is not to commit to infrastructure before you have to. For example, we might wish to write functionality without committing to a particular database. Until a choice is made, we can write a mock class that provides the minimum behavior that we would expect from our database. This means that we can continue writing the tests for our application code without waiting for a working database. The mock code also gives us an initial definition of the functionality we will require from the database.

In the proposed technique, in addition to the Mocks and Fakes, we add the **Test Helpers** technique. Fakes is just replacement of the dependency with the code written to make the test to pass with only the required values. While there are different methods of **Mocking**, where we can use the concepts of C++ for mocking, like:

1. Additional **Constructor** in Class to perform the required functionality.
2. **Templates**: Template specialization can be used to get the required results and happens to be the best technique for mocking and expecting the desired results.
3. **Mocking** a class or a dependent function itself.
4. **Test Helpers:** They provide the technique of testing in all the scenarios and can be used for any number of times and by any anyone.

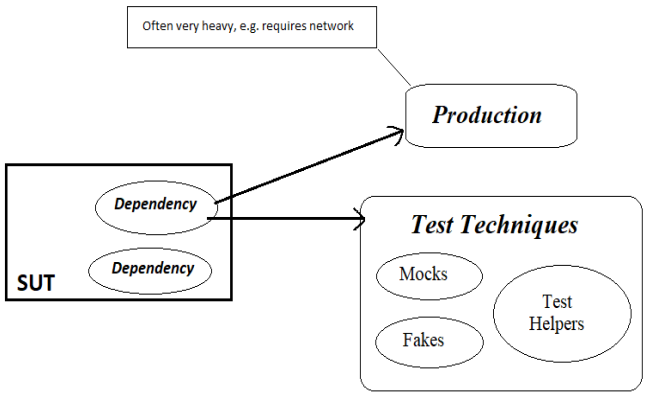

Fig. 3. Proposed Techniques

Test Helpers play a major role in developing the **UT framework** because they provide a virtual platform for testing. If the test requires a huge Database or a Connection, Session or Server details then we can write the Helpers for them. These basically provides the minimum requirement details and perform the very basic functionality in context with the **real world** and give us different types of results by simulating which helps in correct **verification and validation** of the unit of code that is under test. Thus, we get different values from these helpers and at the end our test code is tested, which becomes more error free and the **quality** of the code or the developed application id very high.

Advantages:

1. Deferring Infrastructure Choices.
2. Coping with scale.
3. No stone unturned.
4. Failures fail fast.
5. Easy to learn and understand.
6. Improves the quality of the Application.

V. Conclusion

We have found that Mock Objects and test Helpers is an invaluable technique for developing unit tests. It encourages better-structured tests and reduces the cost of writing stub code, with a common format for unit tests that is easy to learn and understand. It also simplifies debugging by providing tests that detect the exact point of failure at the time a problem occurs. Sometimes, using Mock Objects is the only way to unit test domain code that depends on state that is difficult or impossible to reproduce. Even more importantly, testing with Mock Objects improves domain code by preserving encapsulation, reducing global dependencies, and clarifying the interactions between classes. We have been pleased to notice that colleagues who have also adopted this approach have observed the same qualities in their tests and domain code.

References

[1] S. Ali, L. Briand, H. Hemmati, and R. Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test-case generation. IEEE Transactions on Software Engineering (TSE), 36(6):742–762, 2010.

[2] Andrea Arcuri, Gordon Fraser, Private API Access and Functional Mocking in Automated Unit Test Generation, IEEE, 2017.

[3] Ermira Daka and Gordon Fraser, A Survey on Unit Testing Practices and Problems, IEEE, 2104.

[4] Tim Mackinnon, Steve Freeman, Philip Craig, Endo-Testing: Unit Testing with Mock Objects, 2013.

[5] A. Arcuri, G. Fraser, and J. P. Galeotti. Generating TCP/UDP network data for automated unit test generation. In ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE), pages 155–165. ACM, 2015.