Defects Detection in Imperative Language and C# Applications – Towards Evaluation Approach

S. Sarala

Abstract—**An imperative language such as C++ is a familiar object oriented programming that is widely used for reusability and increased ability to enlighten with other languages. The objective of software testing is to uncover as many errors as possible with a minimum cost. Testing is not confined only to the detection of bugs, it also assists with the evaluation of the functional properties of the software. A piece of software can be tested to increase the confidence by exposing potential flaws or deviations from the user's requirements. Unit testing is to authenticate incorrectness and succeed when an error is detected.**

This work addresses the detection of defects in C# applications, which leads to logical error. Logical errors occur when the code does not perform the way it is intended to perform. The detection and elimination of the logical bug is one of the aims of testing. These errors are very difficult to track since the compiler does not provide assistance. One of the reasons for *Errors* is the presence of unintended characters. A missing or an incorrect piece of code is a defect and it remains undetected until an event activates it. When the code performs unit test, automatically each and individual line of code attempts the syntax checking. It helps to detect and remove all logical and syntactical defects from a given piece of code; unit testing is the most essential technique which can be used for executing the code with checking process. To catch all kinds of defects in the coding phase, the unit tests take a place on C++ and C# applications.

This work includes the defects occur due to unintended characters, wrong usage of data member and formal parameter and a missing argument indicator in console-based applications of C#. In addition to that, the interface anomaly and inheritance anomaly are also detected. Because of enrichment, the comparison has been made on the object oriented source code such as C++ with C# applications. By this approach instantly, unit testing improves the quality of the code in terms of reducing the programmer's burden, time and effort.

Index Terms—**Defects, C++ and C# Programs, inheritance anomaly, interface anomaly and Unit Testing**

I. Introduction

Testing is the process of exercising a program with the intent of finding an error. Software testing helps ensure the quality of the code. Unit testing is testing a subset of the functionality of a piece of software. A unit test is different from a system test in that it provides information. Unit testing is not black box testing, but certain black box tools may be useful to help with monitoring software behavior and creating error conditions. Instead of using unit test, it has some drawback while test the units, the whole structure will not be possible to show for the code. Depends on any language, a unit may be a class, function, procedure or a module. Unit test plays an important role to verify each class's functionality and robustness effectively. It involves simply a code that may starts with a class or function. It also ensures the code's ability to handle unexpected or exceptional situations in the source code.

The unit test is to verify the source code's functionality and construction, also extended to modules, sub-modules and applications. As an individual unit might be tested, the other modules have also been tested. In this work, the unit testing has involved by detecting the interface anomaly and compared the features of C++ and C# applications. Instantly, unit testing improves the quality of the code in terms of reducing the programmer's burden, time and effort.

In this work a few defects have been eliminated that are to prevent the errors by checking the lines of code in C# programs. It is more likely to prove the confidence, reliability and efficiency of the C# applications. Thus, this work takes much effort to compare the concepts of C++ and C# applications. Console based applications like C# are compiled into a stand-alone executable file and run from the command prompt. Input and output is exchanged between the command prompt and the running application. There is no graphical user interface.

II. Unit Testing in Object Oriented Programs

The literature study includes the classes, methods, procedure calls that have been tested through the help of tools [1] [2] [10] [11] [12] [14] [15] [16] by various authors. Unit testing is defined as the smallest compilation unit of the applications [3] [6] [8] [12] [15] [16]. It performs to demonstrate each individual unit such as method, class, objects in the class, procedure call and function call. Various experts have analyses the misused variables, unreachable code and polymorphic faults in the object oriented code.

Tao Xie et.al [23] presented statistical algebraic specifications for identifying special and common object oriented unit tests that are automatically generated tests without requiring specifications. The abstraction is an equation that abstracts the program's runtime behavior like interactions among method calls. They have developed a tool sabicu which was applied for complex data structures.

Mana Taghdiri [9] proposed a new static program analysis method for checking the structural properties of the code. A property is a partial specification of a procedure selected by the user. In this analysis, the original code is finalized by unrolling loops and recursions that constrain the configuration of the heap after the execution of a procedure.

S.Sarala, Lecturer, School of Computer Science and Engineering, Bharathiar University, Coimbatore- 641046, Tamilnadu, INDIA. Phone: 91-0422- 2425743, Mobile: 9444052641; e-mail: sriohmau@ yahoo.co.in.

Yonsik cheon et.al[26] described the tool Junit that attempts to automate unit testing of object oriented programs. They have implemented genetic algorithms for test data generation with the help of Junit. The author peter and parnas [1994] have been developed a tool that generates C++ test oracle procedures from relational program specifications.

Arindam chakrabarthi et.al[25] have identified the control and data inter-dependencies between components by using static program analysis. In their approach the source code are divided into units where highly interwined components which are grouped together. The authors have considered the interfaces of a single unit. The interface of a function describes all possible avenues of exchange of information between the function and its environment arguments return values, shared variables and calls to other functions. The interface of a unit is defined as the interface of the composition of the functions of that unit.

Stefan wappler et.al[24] presented a tree based representation of method call sequences by which sequence feasibility is preserved throughout the entire search process. They applied strongly typed genetic programming that also been employed to generate method call trees for object oriented programs. They have handled runtime exceptions by distance-based fitness function. The detection of defects in C++ and C# programs are progressed by static analysis [17] [18] [19] [20] [21] [22].

B.Y.Tsai et.al, [27] focus on data flow testing. Atif M.Memon et.al, [28] implemented test case generation system called Planning Assisted Tester for graphical user interface systems (PATHS) and experimentally evaluated its practicality and effectiveness. The authors present a new technique to generate test cases automatically by using planning, an Artificial Intelligence Technique.

Jean Hartmann et.al. [29] concentrated on test cases by modelling components using UML state charts. Gregg Rothermel et.al [30] used graph representation for software and used these graphs to select test cases from the original test suite to execute code that has been changed. The authors have worked on regression test selection for C++ software.

Vincenzo Martena et.al [31] automatically produce test case from object-oriented code specifications. They address the problem of interclass testing.

III. Unit Testing on C# Applications

A unit test is a piece of code written by a developer or programmer who exercises a very small, specific area of functionality in the code being tested. Usually a unit test exercises some particular method in a particular context. For instance, the developer or programmer might add a large value to a sorted list and then confirm this value appears at the end of the list or else the developer might delete a pattern of characters from a string and then confirm that they are gone.

A. Introducing C# Applications

C# is a type-safe language for developing enterprise applications. Similarly, the .NET framework provides a run-time environment called the Common Language Runtime that manages the execution of code and provides services for the development process.

The sample of C# program is as follows,

```
class a {
static void main() {
System.Console.WriteLine("Hello");
        }
}
```

Figure 1 Representation of C# structure

The fig 1 shows the string "Hello" on the screen whereas C++ that may recall that the first method. The comparison of C++ programs with C# is very much essential to identify the block of code.

When focus the features of this code, eventually it helps to find the programs dependencies.

- Identifying the individual component/object
- Identifying the method of the class
- Identifying the function of the class
- Identifying the object of the function

Though the method has been invoked in the class during execution the compiler has to perform what the user intended in the code. But it shows an error due to misplace of access. In this work mainly focus on anomaly where the method or function or class is not used in proper way of execution.

In fig 2 depicts the interface anomaly which could try to create an instance of interface and call the collect fee method, then the anomaly "cannot create an instance of the abstract class or interface 'InterfaceDemo.ComputerCenter' is displayed. If an access specifier is used for a method in the interface as,
public void collectFee ( );
then the anomaly the modifier 'public' is not valid for this item.

A defect is an instance in which a requirement is not satisfied. The defects that are incorrect or even missing functionality or specifications may create an unwanted problem. The fig 3 represents the anomaly of accessing the inheritance in C# applications. If the object e is instantiated using default constructor Employee(), then this displays an error. When there is no constructor defined for the class, then the compiler will provide the default constructor. But once the user specifies a constructor for the class, the compiler will not provide the default constructor.

If the statement,

Console.WriteLine("Employee Address: " + Address) in display function of Employee class is changed to the following statement,

Console.WriteLine("Employee Address: " + address) then the anomaly “ inheritanceDemo.Personaddress is inaccessible due to its protection level is displayed. Since the “address” data member is a private member in Person class, it cannot be used in the Employee class.

```
namespace InterfaceDemo
{
class Program
{
 static void Main(string [ ] args)
{
AB s = new AB();
s.Address = “S.S.Puram”;
s.Num_students = 25;
s.collectFee();
Console.Read();
}
}
interface  Computercenter
{
       Void collectFee();
}
class AB :  Computercenter
{
       int num_students;
       public int Num_students
      {
      get {  return num_students;  }
      set  {  num_students  = value;  }
       }
 string address;
 public string Address
{
get {  return address;  }
set  {  address  = value;  }
}
public void collectFee()
{
      Console.WriteLine ( “Inside Collecting fee function”) ;
}
}
}
```

Figure 2 Representing the interface Anomaly of C# Applications

If the “address” member need to be used, then it is possible only through the public setter and getter methods. So the public setter and getter methods for the “address” member in the Person class is “Address”.

The C# sample code in the fig 3 depicts the inheritance anomaly.

```
namespace InheritanceDemo
{
class Program
{
 static void Main(string [ ] args)
 {
      Employee e = new Employee(1, “aaa”, “Chennai”);
      e.display();
      Console.Read();
  }
}
class person
{
      string name;
      public string Name
     {
      get { return name; }
      set { name = value; }
     }
     string address;
     public string Address
    {
    get { return address; }
    set { address = value; }
    }
public Person(string name, string address)
{
   this.name =  name;
   this.address = address;
 } }
class Employee : Person
{
     int empid;
     public int Empid
    {
      get {  return empid;  }
      set  {  empid  = value;  }
    }
    public Employee (int empid, string name, string address) :
    base(name, address)
   {
     This.empid =empid;
   }
   public void display()
  {
     Console.WriteLine(“Empl.Id:” + Empid) ;
     Console.WriteLine(“Em.Name:” + Name) ;
     Console.WriteLine(“Em.Add: ” +Address) ;
 }
}
}
```

Figure 3 Detected code for the inheritance anomaly in C#

IV. Comparison of C++ and C# by Essential Features

In this comparison of C++ with C#, there are no global variables or functions in C# applications. All members and methods must be declared within classes. Unlike C++, local variables cannot shadow variables of the enclosing block. The C# multiple inheritance is not supported, although a class can implement any number of interfaces.

It is more type safe than C++ applications. The novel idea of making comparison is to detect the time consumption and consistency of both applications. This work includes the identification of the anomaly when executing the code of these applications.

Table I The Essential Features of C++ and C# Applications

Category	C++	C#
Data type	int float char	Value-type Reference type Pointer
Predefined value type	struct enum	Struct-type Enum-type
Automatic memory management	Not provided	provided
Garbage Collection	Eligible	Eligible

V. Conclusion

The developed algorithm is to detect the interface anomaly and inheritance anomaly. Some of the important features of the imperative language and C# applications are compared in this work. The aim of work is to detect the unwanted anomaly in the compilers. In this approach the programmer's burden and compiler's efficiency have been checked by means of finding the anomaly where the compiler does not have adequate syntax checking. In terms of applying the unit test, the code's quality has been carried out through the test cases.

References

[1] Arnaud Vernet and Guillaume Brat, 'Precise and Efficient Static Array Bound Checking for Large Embedded C Programs', ACM SIGPLAN Conference on Programming Language Design and Implementation, USA, pp. 1-12.
[2] Atanas Rountev, Ana Milanova and Barbara G. Ryder, 'Fragment Class Analysis for Testing of Polymorphism in Java Software', IEEE Transactions on Software Engineering, Vol.30, No. 6, pp. 372-387.
[3] Beizer B., 'Software Testing Techniques', Van Nostrand Reinhold, New York.
[4] Chi Keen Low and Tsong Yueh Chen, 'CDFA: A Testing System for C++', IEEE Transactions on Software Engineering, Melbourne, pp. 216-228.
[5] David Evans, John Guttag, James Horning and Yang Meng Tan, 'LCLint: A Tool for Using Specifications to Check Code', In The Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 87-96.
[6] Houman Younessi, 'Object-Oriented Defect Management of Software', Prentice Hall, USA.
[7] John Viega J.T., Bloch, Tadayoshi Kohno and Gary McGraw, 'ITS4: A Static Vulnerability Scanner for C and C++ Code', In 16th Annual Computer Security Applications Conference, 2000, pp. 257-267.
[8] Kit Edward, 'Software Testing in the Real World', Addison-Wesley, 1995.
[9] Mana Taghdiri, 'Inferring Specifications to Detect Errors in Code', 19th IEEE International Conference on Automated Software Engineering, 2004, pp. 144-153.
[10] Mark W. Bailey and Jack W. Davidson, 'Automatic Detection and Diagnosis of Faults in Generated Code for Procedure Calls', IEEE Transactions on Software Engineering, 2003, Vol. 29, No. 11, pp.1031-1042.
[11] Misha Zitser, Richard Lippmann and Tim Leek, 'Testing Static Analysis Tools Using Exploitable Buffer Overflows from Open Source Code', In The Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2004, pp. 97-106.
[12] Myers and Glenford J, 'The Art of Software Testing', John-Wiley & Sons.
[13] Nakamura Goichi, Makino Kyoko and Murase Ichiro, 'Buffer-Overflow Detection in C Program by Static Detection', Journal of the National Institute of Information and Communication Technology, 2005, Vol. 52, pp. 35-41.
[14] Ng S.P., Murane T., Reed K., Grant D. and Chen T.Y, 'A Preliminary Survey on Software Testing Practices in Australia', Proceedings of the 2004 Australian Software Engineering Conference, IEEE Computer Society, pp. 116-127.
[15] Ran Patton, 'Software Testing', SAMS Techmedia, 2001.
[16] Robert V. Binder, 'Testing Object-Oriented Systems - Models, Patterns and Tools', Addison Wesley, 1999, pp. 640-641.
[17] Sarala S. and Valli S, 'A Tool to Automatically Detect Defects in C++ programs', In the Proceedings of the 7th International Conference on Information Technology, "Lecture Notes in Computer Science", (LNCS 3356) by Springer-Verlag, 2004, pp. 302-314**.**
[18] Sarala S. and Valli S, 'A Tool to Automatically Generate Test Cases for C++ Programs', In the Proceedings of the 2004 International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, USA, 2004, Vol. 1, pp. 345-351.
[19] Sarala S. and Valli S, 'An Automatic Defect Detection for C++ Programs', Asian International Workshop on Advanced Reliability Modeling, Hiroshima Shudo University, Hiroshima City, Japan, 2004, pp. 419-426.
[20] Sarala S. and Valli S, 'Generation of Test Cases for Console Based Applications in C#', In The Proceedings of the National Conference on Trends of Computational Techniques in Engineering (TCTE), Sant Harchand Singh, Longowal, Central Institute of Engineering and Technology, Longowal, pp. 227-230.
[21] Sarala S. and Valli S, 'Algorithms for Defect Detection in Object Oriented Programs', Information Technology Journal, Asian Network for Scientific Information, Vol. 5, No. 5, pp. 876-883.
[22] Sarala S. and Valli S, 'Algorithms for Defect Detection in Console Based Applications in C#', ICFAI Journal of System Management, ICFAI University Press, India.
[23] TaoXie and DavidNotkin. 'Automatically Identifying Special and Common Unit Tests for Object-Oriented Programs, In Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, Chicago, Illinois, USA, pp. 277-287.
[24] Stefan Wappler and Joachim Wegener., "Evolutionary Unit Testing of Object-Oriented Software Using Strongly-Typed Genetic Programming", The 2006 IEEE Conference on Evolutionary Computation, pp. 3193-3200, Canada.
[25] Arindam Chakrabart and Patrice Godefroid. "Software Partitioning for Effective Automated Unit Testing", Proceedings of the 6th ACM & IEEE International conference on Embedded software, pp.262-271, Seoul, Korea.
[26] Yoonsik Cheon, Myoung Yee Kim and Ashaveena Perumandla, "A Complete Automation of Unit Testing for Java Programs", In Proceedings of the 2005 International Conference on Software Engineering Research and Practice, Nevada, USA, pp. 290–295.
[27] Bor-Yuan Tsai, S.Stobart and N.Parrington, "Employing data flow testing on object-oriented classes", IEE Proceedings on Software Engineering", vol.148, No.2, pp.56-64, April 2001.
[28] Atif M. Memon, Martha E.Pollack and Mary con Soffa, "Hierarchical GUI test Case Generation using Automated

Planning", IEEE Transactions on Software Engineering, vol-27, No.2, pp.144-155, Feb-2001.

[29] Jean Hartmann, Claudio Imoberdorf and Michel Meisinger, "UML-Based Integration Testing", ACM International Symposium on Software Testing and Analysis (ISSTA'00), Portland, pp.60-70, 2000.

[30] G.Rothermel, M.J.Harrold and Jeinay Dedhia, "Regression Test Selection for C++ Software", Journal of Software Testing, Verification and Reliability, Vol.10, No.2, pp.1-35, June-2000.

[31] Vincenzo Martena, Alessandro Orso and Mauro Pezze, "Interclass Testing of Object-Oriented Software", Proceedings of the International Symposium in Software Testing and Analysis (ISSTA'00), IEEE computer society press, pp.10-19, 2000.