Design Integration Based testing using Test Case generation technique

Lalji Prasad[1], Sarita Singh Bhadauria [2]

[1] TRUBA College of Engineering & Technology/ Computer Science &Engineering, INDORE, INDIA
lalji.prasad@trubainstitue.ac.in

[2] MITS /Department of Electronics, GWALIOR, INDIA
saritamits66@yahoo.co.in.

Abstract

In this research work, design an Integration based testing tool (IBTT) based on their properties and it behavior (test cases for Integration based testing) for software development. A requirement specification for an IBT is established that would involve studying the feature set offered by existing software testing tools, along with their limitations. The requirement set thus developed will be capable of overcoming the limitations of the limited feature sets of existing software tools and will also contribute to the design of a Integration based testing tool using class diagram that includes most of the features required for a software testing tool .

Introduction

Object-oriented software testing life cycle development (Establish test objectives, Design criteria or review criteria (Correct, Feasible, Coverage, Demonstrate functionality),Writing test cases, Testing test cases, Execute test cases and Evaluate test results) allows testing to be integrated into each stage of the process from defining requirements to system integration resulting in a smoother development process and a higher end quality.

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing. Below diagram show working process for integration based testing.

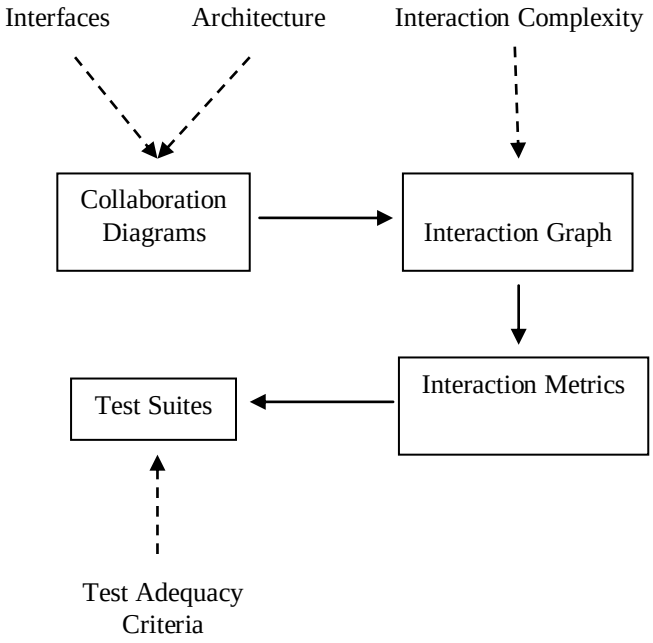

Fig.1.Process for integration testing

The remainder of the paper is organized as follows: Section 2 introduces object-oriented software engineering and presents the various stages of testing. Section 3 discusses the literature survey Section 4 presents the design test cases for IBTT, and section 5 presents the conclusion

2. Object Oriented Testing

Unlike conventional test case design, which is driven by an input-process-output view of software or the algorithmic detail of individual modules, object-oriented testing focuses on designing appropriate sequences of operations to exercise the states of a class. Object-oriented software is developed incrementally with iterative and recursive cycles of planning, analysis, design, implementation and testing .Testing plays a special role here, because it is performed after each increment [19, 20]. Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system conceptual, specification and implementation. These perspectives become evident as the diagram is created and help solidify the design [16].In object oriented perspective Integration testing can be applied in three different incremental strategies:

a) Thread-based testing, which integrates classes required to respond to one input or event.
b) Use-based testing, which integrates classes required by one use case.
c) Cluster testing, which integrates classes required to demonstrate collaboration.

In Integration based testing tool (IBTT) we emphasize above different strategies and there functionality, behavior and relationship.

3. Literature survey and Related work

Generally, we see three major stages in the research and development of testing techniques, each with a different trend. By trend, we mean how mainstream of research and development activities find the problems to solve and how they solve

the problems. As described below, technology evolution involves testing technique technologies. The technique used for selecting test data has progressed from an ad hoc approach, through an implementation-based phase, and is now specification based. The literature survey includes the solution approaches of various research studies that dealt with problems related to testing methods and issues in the design of testing tools for various circumstances.

- **Literature Survey**

[BBL97] A framework for probabilistic functional testing is proposed in this paper. The authors introduce the formulation of the testing activity, which guarantees a certain level of confidence into the correctness of the system under test. They also explain how one can generate appropriate distributions for data domains, including most common domains, such as intervals of integers, unions, Cartesian products, and inductively defined sets. A tool assisting test-case generation according to this theory is proposed. The method is illustrated on a small formal specification [4].

[Beizer90] This book gives a fairly comprehensive overview of software testing that emphasizes formal models for testing. The author provides a general overview of the testing process and the reasons and goals for testing. In the second chapter of this book, the author classifies the different types of bugs that could arise in program development. The notion of path testing, transaction flow graphs, data-flow testing, domain testing, and logic-based testing are introduced in detail. The author also introduces several attempts to quantify program complexity and a more abstract discussion involving paths, regular expressions and syntax testing. The implementation of software testing based on these strategies is also discussed [3].

[BG01] Testing becomes complicated with features, such as the absence of component source code, that are specific to component-based software. This paper proposes a technique combining both black-box and white-box strategies. A graphical representation of component software, called a component-based software flow graph (CBSFG), which visualizes information gathered from both specification and implementation, is described. It can then be used for test-case identification based on well-known structural techniques [5].

[BIMR97] In this paper the authors use formal architectural descriptions (CHAM) to model the behaviors of interest of the systems. A graph of all the possible behaviors of the system in terms of the interactions between its components is derived and further reduced. A suitable set of reduced graphs highlights the specific architectural properties of the system, and can be used for the generation of integration tests according to a coverage strategy, analogous to the control and data flow graphs in structural testing [2].

[GG75] This paper is the first published paper that attempted to provide a theoretical foundation for testing. The "fundamental theorem of testing" proposed by the authors characterizes the properties of a completely effective test selection strategy. The authors argue that a test selection strategy is completely effective if it is guaranteed to discover any error in a program. As an example, the effectiveness of branch and path testing in discovering errors is compared. The use of a decision table (a mixture of requirements and design-based functional testing) as an alternative method is also proposed [7].

[GH88] In this article, the evolution of software test engineering is traced by examining changes in the testing process model and the level of professionalism over the years. Two phase models, the demonstration and destruction models, and two life cycle models, the evolution and prevention models, are provided to characterize the growth of software testing with time. Based on the models, a prevention-oriented testing technology is introduced and analyzed in detail [6].

[Howden76] The reliability of path testing provides an upper bound for the testing of a subset of a program's paths, which is always the case in reality. This paper begins by showing the impossibility of constructing a test strategy that is guaranteed to discover all errors in a program. Three commonly occurring classes of errors, computations, domain, and sub case, are characterized. The reliability properties associated with these errors affect how path testing is defined [9].

[Howden80] The usual practice of functional testing is to identify functions that are implemented by a system or program from requirement specifications. In this paper, the necessity of design testing and requirement functions is discussed. The

paper indicates how systematic design methods, such as structured design and the Jackson design, can be used to construct functional tests. Structured design can be used to identify the design functions that must be tested in the code, while the Jackson method can be used to identify the types of data that should be used to construct tests for those functions [10].

[Huang75] This paper introduces the basic notions of dynamic testing based on a detailed path analysis in which full knowledge of the contents of the source program being tested is used during the testing process. Instead of the common test criteria in which every statement in the program is executed at least once, the author suggested and demonstrated with an example that a better criterion is to require that every edge in the program diagram be exercised at least once. The process of manipulating a program by inserting probes along each segment in the program is suggested in this paper [8].

[Marciniak94] A book intended for software engineers, this book gives introductions, overviews, and technical outlines of the major areas in software engineering. A review of test generators is provided in which the major types of test case generators are given and their intended purpose and principles are discussed. A review of the testing process is given in which the entire process of testing is discussed from planning to execution to achieving to maintenance retesting. All of the common terms and ideas are discussed. A review of testing tools is provided in which the testing tool for each purpose is discussed and several state-of-the-art systems are described [15].

[Miller81] This article serves as one of the introductory sections of the book Tutorial: Software Testing and Validation Techniques. A cross section of program testing technology before and around the year 1980 is provided in this book, including the theoretical foundations of testing tools and techniques for static analysis and dynamic analysis effectiveness assessment management and planning and the research and development of software testing and validation. The article briefly summarizes each of the major sections and provides a general overview of the motivation forces, the philosophy and principles of testing, and the relationship between testing and software engineering [14].

[ROT89] This paper proposes one of the earliest approaches focusing on utilizing specifications in selecting test cases. In traditional specification-based functional testing, test cases are selected by hand based on a requirement specification, which means functional testing merely includes heuristic criteria. Structural testing has an advantage in that the applications can be automated and the satisfaction determined. The authors propose approaches to specification-based testing by extending a wide variety of implementation-based testing techniques to formal specification languages, and they demonstrate these approaches for the Anna and Larch specification languages [17].

[RW85] A family of test data selection criteria based on data flow analysis is defined in this paper. The authors contend that data flow criteria are superior to current path selection criteria in that when using the latter strategy, program errors can go undetected. The definition/use graph is introduced and compared with a program graph based on the same program. The interrelationships between these data flow criteria are also discussed [20].

[Shaw90] Software engineering is still in the process of becoming a true engineering discipline. This article studies the model for the evolution of an engineering discipline and applies it to software technology. Five basic steps are suggested for the software profession in creating a true engineering discipline: understanding the nature of expertise, recognizing different ways to obtain information, encouraging routine practice, expecting professional specializations, and improving the coupling between science and commercial practice. The significant shifts in software engineering research since the 1960s are also discussed in this article [21].

[WC80] Domain errors are in the subset of the program input domain and can be caused by incorrect predicates in branching statements or incorrect computations that affect variables in branching statements. In this paper, a set of constraints under which it is possible to reliably detect domain errors is introduced. The paper develops the idea of linearly bounded domains. The practical limitations of the approach are also discussed, the most severe of which is generating and then developing test points for all boundary segments of all domains of all program paths [23].

[Whit00] As a practical tutorial article, this paper answers questions from developers about how bugs escape testing. Undetected bugs come from executing untested code, differences in the order of executing, combinations of untested input values, and untested operating environments. A four-phase approach is described in answering the questions. By carefully

modeling the software's environment, selecting test scenarios, running and evaluating test scenarios, and measuring testing progress, the author offers testers a structure for the problems they want to solve during each phase [22].

[Poston 2005]
Here we summarize their work-Integration of all the data across tools, repositories and Integration of control across the tools, this Integration to provide a single graphical interface for the test tool set.
Limitation: It emphasizes only integration tools (usability and portability) [19].

[Rosenberg 2008] The approach to software metrics for object-oriented programs must be different from the standard metric sets. Some metrics, such as line of code and cyclomatic complexity, have become accepted as standard for traditional functional/procedural programs. However, for object-oriented scenarios, there are many proposed object-oriented metrics in the literature.
Limitation: This provides only a conceptual framework for measurement [18].

[Anderson 2005] They emphasize that the software industry has performed a significant amount of research on improving software quality using software tools and metrics that will improve the software quality and reduce the overall development time. Good-quality code will also be easier to write, understand, maintain and upgrade [1].

[Lal2011] A full featured comprehensive tool was proposed using the object oriented methodology based architectures [13]

4. Design Class diagram and validation using test case generation for IBTT

In this section of paper we classified Integration testing technique based on their functionality, and validate each member function have set of test cases which are define input or should be identify and corresponding known output should known and method should be trial on these input for its correctness.

For generation of Integration testing test case here we details study of functionality of integration testing follows:
Data member
Private string expected output: The integration testing delivers as its output the integrated system ready for system testing and so the expected output is of string type. The access specifier is kept private so that only the functions of this class can access the output.

Member functions
Public void setup (): Integration testing occurs after class testing. It requires the setup where it takes as input the modules that have been unit tested.

Public void cleanup ():Destroys the unused objects created for performing the testing so that new object will always be used for performing the test on the different piece of data.

Public void runtest (): This method runs the individual software modules.

Public void run (): The group generated after integrating all the software modules is executed as a single unit.

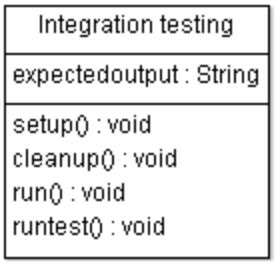

Fig. 4.3: Integration testing

- **Thread based testing**

This type of testing integrates classes required to respond to one input or event. Thread-based integration testing played a key role in the success of this software project. At the lowest level, it provided integrators with better knowledge of the scope of what to test, in effect a contract between developers and testers. At the highest level, it provided a unified status tracking method and facilitated an agreement between management and the developers as to what would be delivered during each formal build.

Data member
Private string expected output: The thread based testing provides with the output of the testing performed on the thread and stores the output in the form of string.

Member function
Public void setup (): This function integrates all the independent threads to verify the functionality of the integration build. Testing is performed after the threads are integrated.

Public void cleanup (): The function judges the status of individual tasks, functional areas and requirements and removes the unwanted and non functional threads.

Public void runtest (): It runs the individual thread in parallel.

Public void run (): It integrates the entire threads and executes them as a single unit.

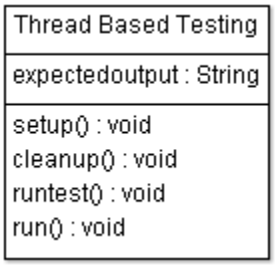

Fig. 4.4: Thread Based Testing

Public void run (): It integrates the entire threads and executes them as a single unit.

```
 #include<stdio.h>
 int calsum(int x,int y,int z);
 int calsub(int p,int q,int r);
 void main()
{
   int a,b,c,result1,result2;
```

```
    int final;
    printf("Enter 3 nos");
    scanf("%d%d%d",&a,&b,&c);
    result1=calsum(a,b,c);
    result2= calsub(p,q,r);
    final= result1* result2;          // integrates the result of two independent functions.

}

int calsum(int x,int y,int z)
{
     int d;
     d=x+y+z;
     return(d);
}
int calsub(int p,int q,int r)
{
     int d;
     d=x-y-z;
     return(d);
}
```

Void run	Property
The function integrates all the independent threads	The function integrates the result obtained by the two functions calsum() and calsub() and integrates the result of both to obtain the final result.

Public void run test (): It runs the individual thread in the order they are called.

```
#include<conio.h>
void main()
{

alpha();                                   // Calling of function alpha()
printf("Software engineering");
```

```
getch();
}

alpha()

{

printf("testing");
beeta();                                    // Calling of function beeta()
printf("Object oriented testing");

}
beeta()
{
printf("Mumbai);
```

Function called	Function executed	Property
alpha,beeta	alpha,beeta	The function called are being executed in order in which they are called.

- **Cluster based testing**

This type of testing integrates classes required to demonstrate collaboration. Cluster testing is one step in the integration testing of OO software. Here, a cluster of collaborating classes is determined by examining the CRC and object-relationship model.

Data member
Private string expected output: It stores the output of the similar type of classes i.e. clusters in the form of string.

Member function
Public void setup (): It identifies the similar classes and groups them into a cluster which is ready to be tested.

Public void cleanup (): It destroys all the clusters created during the entire testing.

Public void run (): It executes the cluster generated for testing.

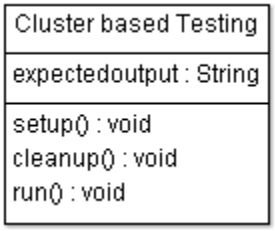

Fig. 4.5: Cluster Based Testing

- **Use based testing**

This type of testing integrates classes required by one use case. Use based testing implies a focus on detecting faults that cause the most frequent failures i.e. User-friendliness check. Unlike structural code based testing, no prior knowledge of the program is necessary for use based testing. Application flow is tested here. So the testers and the programmers can be independent of one another. However, the testers are required to understand the requirement specifications, and they should be familiar with how the software behaves in response to the particular action.

Data member
Private string expected output: The testing emphasizes on detecting faults that cause the most frequent failures and so the output is kept of type string. To provide only the class accessibility it is kept private.

Member function
Public void setup (): In use based testing the testers are required to understand the requirement specifications, and so the function provides the user with initial requirements.

Public void cleanup (): It destroys all the use cases after the testing is performed.

Public void use cases (): The used based testing integrates classes required by one use case and so this function provides with the integrated classes required.

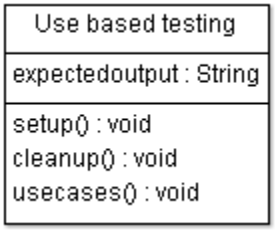

Fig. 4.6: Use Based testing

5. Conclusion

This Architecture testing tool (IBTT) solves numerous problems for effective Integration based testing. Development of efficient testing techniques and tools that will assist in the creation of high-quality software will become one of the most important research areas in the near future. IBTT test whose success implies Program Correctness and increase software reliability. These designs can be further translated into practical industrial tools.

6. References

[1]. Anderson John L. Jr., "How to Produce Better Quality Test Software", IEEE Instrumentation & Measurement Magazine, August 2005.

[2]. Bertolino A., Inverardi `P., Muccini H. and Rosetti A., "An approach to integration testing based on architectural descriptions," Proceedings of the IEEE ICECCS- 97, pp. 77-84.

[3]. Beizer B., "Software Testing Techniques," Second Edition, Van Nostrand Reinhold Company Limited, 1990, ISBN 0-442-20672-0.

[4]. Bernet G., Bouaziz L. and LeGall P., "A Theory of Probabilistic Functional Testing," Proceedings of the 1997 International Conference on Software Engineering, 1997, pp. 216 –226.

[5]. Beydeda S. and Gruhn V., "An integrated testing technique for component-based software," ACS/IEEE International Conference on Computer Systems and Applications, June 2001, pp 328 – 334.

[6]. Gelperin D. and Hetzel B., "The Growth of Software Testing", Communications of the ACM, Volume 31 Issue 6, June 1988, pp. 687-695.

[7]. Goodenough J.B. and Gerhart L., "Toward a Theory of Test Data Selection," IEEE Transactions on Software Engineering, June 1975, pp. 156-173.

[8]. Huang J. C., "An Approach to Program Testing," ACM Computing Surveys, September 1975, pp.113-128.

[9]. Howden W. E., "Reliability of the Path Analysis Testing Strategy", IEEE Transactions on Software Testing, September 1976, pp. 208-215.

[10]. Howden W. E., "Functional Testing and Design Abstractions", the Journal of System and Software, Volume 1, 1980, pp. 307-313.

[11]. Miller E. F., "Introduction to Software Testing Technology", Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO 180-0, pp. 4-16.

[12]. Marciniak J. J., "Encyclopedia of software engineering", Volume 2, New York, NY: Wiley, 1994, pp.1327-1358.

[13]. Prasad Lalji. Bhadauria S. and Kothari A., "A full featured component object oriented based architecture testing tool" ,IJCSI Sep2011

[14]. Richardson D., O'Malley O. and Title C., "Approaches to specification-based testing", ACM SIGSOFT Software Engineering Notes, Volume 14 , Issue 9, 1989, pp. 86 – 96.

[15]. Rosenberg Linda H., "Applying & interpreting object oriented Metrics", 2008.

[16]. Robert M. Poston, "Testing tool combines best of new and old", IEEE Software. March 2005.

[17]. Rapps S. and Weyuker E. J., "Selecting Software Test Data Using Data Flow Information", IEEE Transactions on Software Engineering, April 1985, pp. 367-375.

[18]. Shaw M., "Prospects for an engineering discipline of software", IEEE Software, November 1990, pp.15-24.

[19]. Whittaker J. A., "What is Software Testing? And Why Is It So Hard?" IEEE Software, January 2000, pp. 70-79.