A Model (Software Testing System) for Software specification and testing using Object – Oriented DBMS.

Ashvini Shende
Asst.Professor,
Indira College of Commerce & Science,Pune
ashwinishende@iccs.ac.in

Sarita Byagar
Asst.Professor,
Indira College of Commerce & Science,Pune
sarita.byagar@iccs.ac.in

Mahesh Jagtap
Asst.Professor,
Indira College of Commerce & Science,Pune
Mahesh.jagtap@iccs.ac.in

***Abstract* - This Paper will describe use of object oriented database in software testing. Software Testing System (STS) will store structural and functional description of software requirements, software system, test data and their expected result, bug reports to be generated and stored in software database using object oriented database system. Initially this model will focus on manual testing.**

***Keywords*: Sotfware Testing,Manual Testing,OODBMS**

INTRODUCTION

This paper will focus on use of Software Testing System Model in manual testing using Object Oriented Database Concept.
This Paper is of 4 Sections, first section gives Introduction of keywords, second section focuses on literature review, third section briefs us about STS model for manual testing, and last section will conclude for the objective.
The field of software engineering is concerned with delivering quality software, within budget and schedule constraints, that satisfies the needs of the client and user.
Software bugs will almost always exist in any software module with moderate size: not because programmers are careless or irresponsible, but because the complexity of software is generally intractable -- and humans have only limited ability to manage complexity. It is also true that for any complex systems, design defects can never be completely ruled out.
Software Testing is the process of executing a program or system with the intent of finding errors [1].
When a program is implemented to provide a concrete representation of an algorithm, the developers of this program are naturally concerned with the correctness and performance of the implementation. Software engineers must ensure that their software systems achieve an appropriate level of quality. Software verification is the process of ensuring that a program meets its intended specification [2]

There are an abundance of software testing tools exist. The correctness testing tools are often specialized to certain systems and have limited ability and generality. Robustness and stress testing tools are more likely to be made generic. Mothora is an automated mutation testing tool-set developed at Purdue University.[3]
Using Mothora, the tester can create and execute test cases, measure test case adequacy, determine input-output correctness, locate and remove faults or bugs, and control and document the test. NuMega's Boundschecker [5,6] They are run-time checking and debugging aids. They can both check and protect against memory leaks and pointer problems.
Ballista COTS Software Robustness Testing Harness [4]. The Ballista testing harness is
an full-scale automated robustness testing tool. The first version supports testing up to 233 POSIX function calls in UNIX operating systems. The second version also supports testing of user functions provided that the data types are recognized by the testing server. The Ballista testing harness gives quantitative measures of robustness comparisons across operating systems. The goal is to automatically test and harden Commercial Off-The-Shelf (COTS) software against robustness failures.Regardless of the limitations, testing is an integral part in software development. It is broadly deployed in every phase in the software development cycle. Typically, more than 50% percent of the development time is spent in testing. Testing is usually performed for the following purposes:

- **To improve quality.**

As computers and software are used in critical applications, the outcome of a bug can be severe. Bugs can cause huge losses. Bugs in critical systems have caused airplane crashes, allowed space shuttle missions to go awry, halted trading on the stock market, and worse. Bugs can kill. Bugs can cause disasters. The so-called year 2000 (Y2K) bug has given birth to a cottage industry of consultants and programming tools dedicated to making sure the modern world doesn't come to a screeching halt on the first day of the next century. In a computerized embedded world, the quality and reliability of software is a matter of life and death.

Quality means the conformance to the specified design requirement. Being correct, the minimum requirement of quality, means performing as required under specified circumstances. Debugging, a narrow view of software testing, is performed heavily to find out design defects by the programmer. The imperfection of human nature makes it almost impossible to make a moderately complex program correct the first time. Finding the problems and get them fixed [2], is the purpose of debugging in programming phase.

- **For Verification & Validation (V&V)**

Just as topic Verification and Validation indicated, another important purpose of testing is verification and validation (V&V). Testing can serve as metrics. It is heavily used as a tool in the V&V process. Testers can make claims based on interpretations of the testing results, which either the product works under certain situations, or it does not work. We can also compare the quality among different products under the same specification, based on results from the same test.

We can not test quality directly, but we can test related factors to make quality visible. Quality has three sets of factors -- functionality, engineering, and adaptability. These three sets of factors can be thought of as dimensions in the software quality space. Each dimension may be broken down into its component factors and considerations at successively lower levels of detail. Table 1 illustrates some of the most frequently cited quality considerations.

Functionality (exterior quality)	**Engineering (interior quality)**	**Adaptability (future quality)**
Correctness	Efficiency	Flexibility
Reliability	Testability	Reusability
Usability	Documentation	Maintainability
Integrity	Structure	

Table 1. Typical Software Quality Factors [Hetzel88]

Good testing provides measures for all relevant factors. The importance of any particular factor varies from application to application. Any system where human lives are at stake must place extreme emphasis on reliability and integrity. In the typical business system usability and maintainability are the key factors, while for a one-time scientific program neither may be significant. Our testing, to be fully effective, must be geared to measuring each relevant factor and thus forcing quality to become tangible and visible. [7]

Tests with the purpose of validating the product works are named clean tests, or positive tests. The drawbacks are that it can only validate that the software works for the specified test cases. A finite number of tests cannot validate that the software works for all situations. On the contrary, only one failed test is sufficient enough to show that the software does not work. Dirty tests, or negative tests, refer to the tests aiming at breaking the software, or showing that it does not work. A piece of software must have sufficient exception handling capabilities to survive a significant level of dirty tests.

A testable design is a design that can be easily validated, falsified and maintained. Because testing is a rigorous effort and requires significant time and cost, design for testability is also an important design rule for software development.

These objectives are achieved by the following test activities during software life-cycle:
1. Test requirements and definitions (SQA plan)
2. Test plan development
3. White box testing
4. Black box testing
5. Independent testing (Alpha & Beta testing)
6. Regression testing
8. Metrics and measurements

It is strongly recommended that testing should start early since the later a software defect is discovered the more costly it is to fix the problem [8]. Although the testing phases are listed after the coding and debugging phases of software development process, the actual testing activities should begin at the design phase of the lifecycle
process. Software testers should begin test suite design early in the process in order to build quality and testability into the design of the product and to develop test plans and test specifications for the test suite development. Early involvement of software testers has many benefits, including familiarity with requirements and design philosophy of the product.
The traditional approach to implementing testing activities described here are both time consuming and expensive. The primary reasons why testing is so time consuming and expensive is:
1. Lack of facilities to allow local testing
2. Lack of adequate integration of testing facilities

3. Inflexible test suite maintenance system
4. Lack of facilities for information sharing and coordination among the various groups

An object oriented approach to the development of software was first proposed in the late 1960s. Now, the object oriented technologies have slowly replaced classical software development and database design approaches.

Object Oriented databases have adopted many of the concepts that were developed originally for object-oriented programming languages.

An object-oriented database (OODB) is a persistent and sharable collection of objects defined by an Object Oriented Model. An OODB can extend the existence of objects so that they are stored permanently. Hence, the objects persist beyond termination and can be retrieved later and shared by other programs.

Their characteristics include, maintaining a direct correspondence between real-world and database objects to preserve objects integrity and identity, support for OOP concepts viz encapsulation, inheritance etc, capable of defining new data types as well as the operations to be performed on them i.e extensible and many more[9]

LITERATURE SURVEY

For writing this paper done analysis and gathered some failure cases of softwares., which are listed below:

1. **Forty seconds of Ariane-5**

The European Space Agency (ESA) has also suffered embarrassment on the software front. The inaugural flight of its fifth-generation Ariane launcher bested NASA's Mariner 1 score for unmanned spacecraft disaster: It took only 40 seconds to blow up.

On June 4, 1996, after the kind of dramatic vertical blastoff you'd expect from a high-profile European vehicle, cameras on the ground barely had time to focus on the Ariane-5 as it turned around and began to fall apart, before it completely exploded.

The Ariane Flight 501 disaster began with a loss of guidance and attitude information 30 seconds after liftoff. Once it veered completely off course, it automatically self-destructed. The problem was that Ariane-5's inertial reference system dealt with 64-bit floating-point data and converted it into 16-bit signed integer values. The result of the data conversion was too large for a 16-bit signed integer, which caused an arithmetic overflow in the hardware. In the ESA's case, a software handler that could have dealt with the problem had been disabled, and so there was no levee to dam the cascade of system failures that led to the destruction.
[10]

The bug that never was: Black Monday's dark secret

It is a truth universally acknowledged (by people who don't know bugs) that the end of the 1980s stock boom,Black Monday of 1987, was precipitated by buggy software. It was Wall Street's greatest ever loss in a single day: The Dow Jones Industrial Average plummeted 508 points, 22.6% of its total value, and the S&P 500 dropped 20.4%. And it was all the fault of bugs in the computer models.

Except that it wasn't.

Program trading was relatively new and harder to understand back then, and people with diminished pension funds were anxious to find a scapegoat they could really lay the blame on. It was easier to point to a faulty program than to understand overvaluation, lack of liquidity, international disputes about exchange rates, and the market's notoriously bipolar psychology. So the computers became the bad guys.

Of course, program trading did contribute to the precipitous fall of American markets. The software contained strategy models for handling portfolio insurance, and it was there that the problems of Monday, Oct. 19, 1987, really lay. Portfolio insurance derivatives are tied to the condition of the market. After things nose-dived in Hong Kong and Europe, the sun rose on a Wall Street ready to react: The writers of derivatives sold on every down-tick, and plummeting values triggered a cascade of selling.

But the trading programs just did as they were instructed. The fact that they sold as the financial markets collapsed around them wasn't a bug, it was a feature -- just not a well-thought-out one.[11]

Objective & Methodology

Our objective is to develop a Software Testing System (STS) Model which supports various testing activities through a high-level and uniform user interface. The testing facilities which are provided integrate three domains: the Requirement Specification domain, the software test domain, and the test execution domain.

1. The Requirement Specification Domain

First Domain provide facilities that allow
“structural” and “functional” representation of software to be produced, maintained, retrieved, and displayed. Software application system is considered to be an aggregation of programs that are themselves aggregations of source files.

A source file is defined to be a textual representation of requirements. A file is further defined to include one or many “structural-units.”

From a testing perspective, a structural-unit is a segment of consecutive lines of specification from modules.

Characteristics of requirement specification domain:
1. It is self contained with only one entry and one exit.However, it can contain control-transfer statements within itself.
2. It is associated with one or more unit tests
3. It performs an identifiable task

Structural representation of software views the
software product from its building block perspective, and the functional representation views it from its functionality and

usage perspective. The later representation examines how the software product is utilized by the end users and how closely it conforms to user requirements. Software functionality is identified by tracing the execution path through structural-units of the software. From a testing
perspective, functional testing implies verification of
selected execution paths (complete verification of all
execution paths is impractical). At the lowest level of
functionality structural testing and functional testing
converge with both methods verifying correctness of the structural-units.

The functionality of a software product is described in the functional specification document. This document is he guiding document for several groups. The developers use this document **as** the basis of their design; the technical publication group uses it for developing user manuals and reference guides, and the software quality assurance group
uses it as the basis for developing manual test plan.

2. Software Test Domain

The STS provides facilities that allow test tools, test
Test data, and test documents are stored in OODBMS format.
underlying object-oriented database system. The
database includes test plans, test specifications, test suites,test data, and expected results. The STS also provides a test database maintenance, which includes test template

and the requirement specification domain. These facilities also provide capabilities allowing mapping between the test case and the structural representation of the software system, the test case and the functional specification of the software system, and the test case and expected results of the execution of the test case.

3. Test Execution Domain

The STS provides facilities to support test execution
and analysis using information and tools from both the requirement specification and the software test domains.

1. The STS will provide tools to support manual testing.
2. Facilities are provided for checking bug report from database structure, or execution path.
3. Defects that are the results are stored with relative information of its corresponding module.

.

generation, defect classification and test result report generation.

Test suites are partitioned into unit tests, module/
function tests, and system or integration tests. Each test case includes three separate sections. The first section is a header section representing a textual description of the test case, its author and other relevant information. The second section includes the actual test data for the test case. Finally, the last section includes the expected results of the test case.

The unit testing **as** defined here tests the correctness of **a** task performed by a structural-unit and, therefore, corresponds to the functional testing of the software at the lowest level of functionality. Complete structural testing of the software application system can thus be performed
by executing all such unit tests.

Once unit testing is successfully completed, functional and system/integration testing begins. Functional testing of the software application system consists of two complementary stages:

1. In the first stage, test data is used to validate correct implementation of every software feature **as**
defined by the functional specification.
2. The second stage includes functional testing of the
software product at the system level and as a whole.

The STS provides facilities that allow mappings among the components of the software test domain

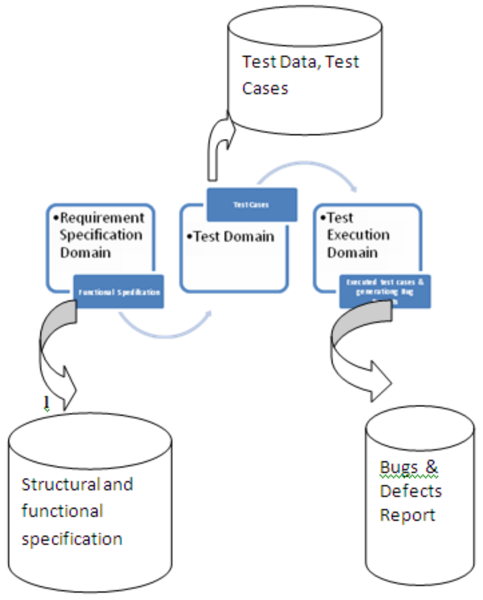

Fig. Software Testing System Model

CONCLUSION

The above model will helps testers & engineers to understand requirements and will help to minimize

the errors. Easy to trace errors from respective modules with the help of OODBMS.
Requirements are stored in the form of functional specification model which relates with other domain and gives test data & test cases for efficient testing.
Reduce the time and cost of manual testing.

REFERENCES

[1] [Myers79] Myers, Glenford J., The art of software testing, Publication info: New York : Wiley, c1979. ISBN: 0471043281 Physical description: xi, 177 p. : ill. ; 24 cm.
[2] [Kaner93] Cem Kaner, Testing Computer Software. 1993.
[3] Database Systems – Concepts, Designs and Application by Shio Kumar Singh, second edition
[4] http://www.cs.tau.ac.il/software hororstories
[5] [DeMillo91] Richard A. DeMillo. ICSE, page 180-183. IEEE Computer Society / ACM Press, (1991)
[6] [Ballista99] http://www.cs.cmu.edu/afs/cs/project/edrc-ballista
[7] [NuMega99] NuMega's Boundschecker .They are run-time checking and debugging aids.
[8] [Rational99] .Rational's Purify .They are run-time checking and debugging aids.
[9] [Hetzel88] Hetzel, William C., The Complete Guide to Software Testing, 2nd ed. Publication info: Wellesley,Mass. : QED Information Sciences,1988. ISBN: 0894352423.Physical description: ix, 280 p. : ill ; 24 cm.
[10] Ghiassi, M. "Software Quality: Design It in from the Start," "Computer Design" (1984), Vol. 23, No.9, pp.91-97
[11] 11. www.computerworld.com