Software Testing Strategies and Current Issues in Embedded Software Systems

Sheena Singh, Amandeep Kaur, Kapil Sharma, Saurabh Srivastava

Abstract: software testing is a very important phase for checking the correctness and achieving reliability measures of a product. Once software is completed it is passed through testing process, now days, some life cycle models continuously test the software throughout the development phase. Two major testing techniques are Black-Box and other one White-Box testing. In this paper we compare few pre-existing black box and white box testing process with a new white box testing technique implemented using ant colony optimization algorithm and present some real life examples of software bugs that are found in embedded systems.

Index Terms: Software testing, Functional testing, Structural testing, test cases, black box testing, white box testing, Testing techniques.

1 *Introduction:*

Software testing is used for checking the correctness of software. A software must be a bug free software if we want to prevent loss of life or resources. A billion $ satellite launch project was turned into ashes because of a software bug. Thus testing of each and every module is necessary. Testing can be performed either in black box way or white box also known as functional and structural testing respectively. With the intent of finding software errors so that the errors can be corrected before release of software to the end users the testing is performed.

Why testing is important?

On April 26 1994 Chinese airplane, airbus 8300 crashed killing 264 people, similarly a Canadian therac-25 radiation therapy machine malfunctioned due to software bug and increased radiation causes 3 people dead and 3 others critically injured. In may 1996 a U.S. bank bear a loss of $9.2 billion as funds were transferred to 823 customers of that bank. There are many more similar cases like mentioned above which are live example of damages caused by a simple software bug.

Testing principles

There are seven major testing principles which must be considered while testing a product. All these principles collectively tell us about the pros and cons of the process of testing.

- Testing shows the presence of defects
- Exhaustive testing is impossible
- Early testing must be done
- Defect clustering
- Pesticide paradox
- Testing is context-dependent
- Absence of error is a fallacy

First point is very usual as we need to find bugs in software thus we test it. Now suppose if we have 10 fields to be filled with 5 different inputs then total no. of test cases will be as large as

5^10 = = billions (approx.)

Thus, exhaustive testing is quite impossible as it will take lot of effort and resource to test all the possible combinations within certain time limit.
Early testing will provide you clear idea about the software and bugs can be detected for every simplest module of a software product.
Defect clustering can be done from testers experience as software may have many bugs in a single module which can be clustered out to be tested.
But, pesticide paradox is something which says that if you are using same set of test cases again and again then chances of finding defects reduces, thus we need to regularly check and modify our test cases on regular basis.
Even after all this effort you can never claim your software as bug-free software. On the launch of windows 98 in 1998 during the public demonstration windows 98 was crashed which was the first operating system designed for public use including web-friendly elements. Thus, testing reduces the probability of undiscovered defects in any software.
Testing is context dependent means; each software is tested in different manner i.e. an e-commerce site will be tested in different manner than any social networking site.
Each and every testing method has its some own advantages and disadvantages. Sometimes there are bugs that cannot be found using black box or only white box alone.
Majority of the applications are tested by black box testing method. We need to cover a majority of test cases so that most of the bugs get discovered by black-box testing.
Black box testing occurs throughout the software development process i.e. in Unit, Integration, System, Acceptance and regression testing stages.

- *Sheena Singh is currently teaching as an assistant professor in computer science and engineering department at lovely professional university, Country, PH-9646689681. E-mail: sheena.15740@lpu.co.in*
- *Amandeep Kaur is currently pursuing masters degree program in computer science and engineering department at lovely professional university, Country, PH-9465080714. E-mail:damandep4@gmail.com*
- *Saurabh Srivastava is currently pursuing M.tech in computer science and engineering department from lovely professional university, India, PH-8699499269. E-mail:iam100rabh@gmail.com*
- Kapil Sharma is currently pursuing M.tech *in computer science and engineering department from lovely professional university, India, PH-8283809270. E-mail: kapilsharma701@gmail.com*

Tools used for Black Box testing:

Black box testing tools are mainly record and playback tools. These tools are used for regression testing that to check whether new build has created any bug in previous working application functionality. These record and playback tools records test cases in the form of some scripts like TSL, VB script, Java script, Perl etc.

Similar to black box testing there are several white-box testing methods, which are used to test the internal functional or underlying code structure of the software. White-box testing is also known as glass-box testing or structural testing. The various white-box testing techniques are listed below.

- Statement coverage
- Branch coverage
- Path coverage
- Condition coverage
- Function coverage
- Multiple condition coverage
- Basis path testing
- Flow based testing

Tools for white-box testing

Cantata++ can be used for statement coverage; TCAT-PATH can be used for branch coverage, similarly there are few more software that provide above mention testing facilities.

x-suds is a software tool to perform basis path testing, this is one of the most important types of software testing. It requires control flow graph. The main focus in this testing is to write test case in such a way that it covers all possible feasible paths including each and every node and edges. We make use of CFG in BPT and we calculate Cyclomatic complexity in order to get all the possible paths in CFG from start to end node.

McCabe introduced the concept of basis path testing in 1980's which make use of Cyclomatic complexity. But by just calculating possible paths doesn't ease our work as there are number of infeasible paths in the control flow graph.

So we need to tackle with infeasible paths if we want our algorithm to be efficient and thus we can introduce the concept of ant colony optimization. This is a very majorly used approach for prioritization of path, as we can see many applications of ACO, like, travelling salesman problem using ant colony optimization, ant colony optimization algorithm is a meta-heuristic approach which is based on the activity of ants, i.e. how ants choose a path while taking their food to their colony.

Formula for Ant colony optimization algorithm can be given by:

$$Pij = \frac{((Peij)^X * (Vpij)^y)}{\Sigma ((Peij)^x * (Vpij)^y)}$$

1) Past Experience or pheromone value (Pe).

2) Visibility of path or heuristic value (Vp).

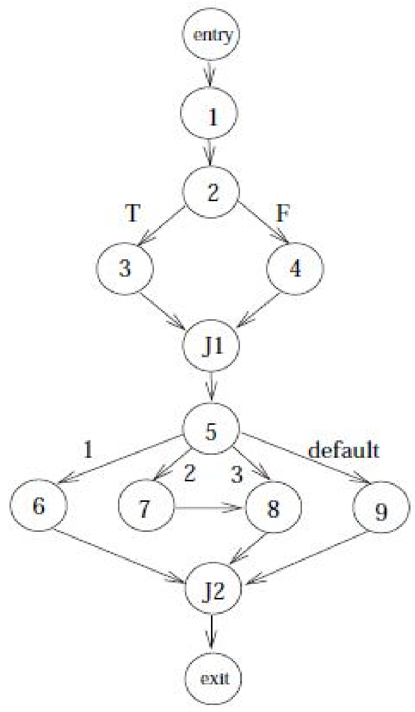
3) x and y are desirability and visibility factors.

This paper also presents an extended approach of ant optimization algorithm which shows that chances of path selection of shorter paths increases. Tables are showing the improved chances of path selection.

Let us take an example, using ACO in BPT on a program trivial:

Program Trivial

```
1. read(n);
2. if (n<0) then
3. write("negative");
        else
4. write("positive");
        endif;
5. switch (n)
        case 1:
6. write("one");
7. write("two");
        break;
        case 2:
        case 3:
8. write("three");
        break;
9. write("other");
        default:
        endswitch;
```



Control flow graph for program trivial

P1→ 1-2-4-J1-5-6-J2-exit

P2→ 1-2-4-J1-5-7-8-J2-exit

P3→ 1-2-4-J1-5-8-J2-exit

P4→ 1-2-4-J1-5-9-J2-exit

P5→ 1-2-3-J1-5-6-J2-exit

P6→ 1-2-3-J1-5-7-8-J2-exit

P7→ 1-2-3-J1-5-8-J2-exit

P8→ 1-2-3-J1-5-9-J2-exit

These are eight possible paths which can be derives from control flow graph but out of these P1,P2,P3,P4 are only feasible paths, rest are infeasible paths, because they cannot be executed using any set of test case.

path	nodes traversed	ΣPij	priority
P1	1-2-4-J1-5-6-J2-exit	6.75	4
P2	1-2-4-J1-5-7-8-J2-exit	7.75	1
P3	1-2-4-J1-5-8-J2-exit	7.00	3
P4	1-2-4-J1-5-9-J2-exit	7.50	2
P5	1-2-3-J1-5-6-J2-exit	-------	------
P6	1-2-3-J1-5-7-8-J2-exit	-------	------
P7	1-2-3-J1-5-8-J2-exit	-------	------

Table of path prioritization using ACO algorithm in BPT

Enhanced formula given as:

Cumulative probability = Pij / max (Pij)

path	nodes traversed	Σpij/max Pij	priority
P1	1-2-4-J1-5-6-J2-exit	0.75	4
P2	1-2-4-J1-5-7-8-J2-exit	0.775	3
P3	1-2-4-J1-5-8-J2-exit	0.777	2
P4	1-2-4-J1-5-9-J2-exit	0.83	1
P5	1-2-3-J1-5-6-J2-exit	-------	------
P6	1-2-3-J1-5-7-8-J2-exit	-------	------
P7	1-2-3-J1-5-8-J2-exit	-------	------

Table of path prioritization using extended ACO in BPT

Blank portion in table represents infeasible paths. The above table demonstrates the paths with shorter length or say paths with less number of nodes are having more probability of selection thus a tester will find the above approach more feasible to apply than previous case.

Types of Testing:

Testing is performed on each kind of software, i.e. either the software is generic, customized or embedded. In this paper we present some test results after testing embedded software, before moving onto analysis of embedded software systems, we define types of software

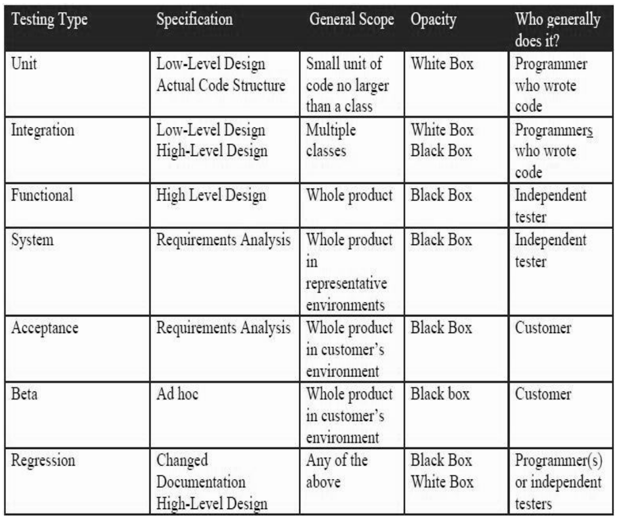
Generic: it is software for all, i.e. these software are available for everyone. E.g. free software open licensed.

Customized: it is build by focussing on target users e.g. web sites, desktop applications etc.

Embedded: these are those software which can't be modified once installed on hardware system and has some limited functionality. E.g. washing machines, microwaves, mobile phones etc.

Now we discuss about different testing techniques from bottom to top level. Generally we start from unit testing which is the most basic testing technique in which we test the most basic module of a software. Again, integration testing is performed after unit testing , then functional system testing is performed. Acceptance testing is done by user or customer. All this can be understood using the table below.

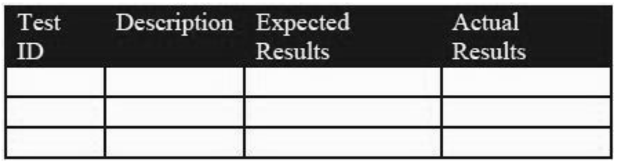
levels of testing are:



Testing Type	Specification	General Scope	Opacity	Who generally does it?
Unit	Low-Level Design Actual Code Structure	Small unit of code no larger than a class	White Box	Programmer who wrote code
Integration	Low-Level Design High-Level Design	Multiple classes	White Box Black Box	Programmers who wrote code
Functional	High Level Design	Whole product	Black Box	Independent tester
System	Requirements Analysis	Whole product in representative environments	Black Box	Independent tester
Acceptance	Requirements Analysis	Whole product in customer's environment	Black Box	Customer
Beta	Ad hoc	Whole product in customer's environment	Black box	Customer
Regression	Changed Documentation High-Level Design	Any of the above	Black Box White Box	Programmer(s) or independent testers

This table can be described as levels of software testing i.e. from unit to regression testing.

Also for testing we need to design test cases, test cases planning template can be shown like,



Test ID	Description	Expected Results	Actual Results

Now we discuss some problems that are found in real time scenario in embedded systems in present time and will provide a likely cause of it.

Embedded systems are those systems which are targeted for a particular machine and cannot be modified by end users, only it can be used upto intended level. E.g. mobile phones, in present trend mobile phone companies have started producing mobile phones at very cheap rate. It must be noted that "testing of software" is one of the most expensive task while developing software, thus, cheaper devices are not readily tested. That's why most of the time the mobile phones software does not work correctly and its software fails, in general term we say it "hangs".

Apart from Nokia, erricson, Samsung, and few other companies there are many new companies that entered into the market of developing mobile phones some are Micromax, Karbonn, Celkon, Spice, Lemon etc. These companies are providing phones at cheaper rate than above listed companies. These new companies build the software and install it on hardware but they really don't put their emphasis on testing the software.

There are many major problems that were reported with many models of their phone.

One model of Micromax have few bugs that were reported which are listed below.

- Throws exception while typing any number starting from 97, in this case one cannot dial a no. which starts with 97 and so on.
- In its dual sim series, it does not work properly when only one sim is inserted, i.e. phone hangs and process very slowly when only one sim card is inserted.
- While charging the phone, one cannot type message, a sequence of random characters display when you press any key in message box etc.

Similarly other phones of other companies were also reported. These are due to software failure, because software were not tested thoroughly the bugs remains which cause problems to end users and also cause loss to reputation of a company.

In the table described above as levels of software testing, it shows functional testing must be done by independent testers. Thus above problems can be solved using independent testers.

Conclusion:

This paper defines a complete set of both ways of testing, namely black and white box testing. It also compares the two white box testing technique i.e. basis path testing, one with ACO and other with extended ACO. The paper also gives you idea about why the software failure occurs in any embedded system. We must emphasise on testing while developing a software as it can result harmful to man and resources as discussed earlier.

References:

[1] *P. R. Srivastava "An Approach of Optimal Path Generation using Ant Colony Optimization" IEEE TENCON 2009*

[2] *Path Testing Mohammad Mousavi Eindhoven University of Technology, The Netherlands Software Testing, 2012*

[3] *"A path-oriented automatic random testing based on double constraint propagation" Ruilian Zhao, Yuandong Huang International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.2, March 2012*

[4] *black box and white box testing techniques –a literature review, Srinivas Nidhra and Jagruthi Dondeti, International Journal of Embedded Systems and Applications (IJESA) Vol.2, No.2, June 2012*

[5] *White Box Coverage and Control Flow Graphs Venezia Elodie, 2011*

[6] *L. Copeland, A Practitioner's Guide to Software Test Design. Boston: ArtechHouse Publishers, 2004.*

[7] *R. D. Craig and S. P. Jaskiel, Systematic Software Testing. Norwood, MA: Artech House Publishers, 2002.*

[8] *E. W. Dijkstra, "Notes on Structured Programming," Technological University Eindhoven T.H. Report 70-WSK-03, Second edition, April 1970.*

[9] *D. Galin, Software Quality Assurance. Harlow, England: Pearson, Addison Wesley, 2004.*

[10] *IEEE, "ANSI/IEEE Standard 1008-1987, IEEE Standard for Software Unit Testing," no., 1986.*

[11] *IEEE, "ANSI/IEEE Standard 1008-1987, IEEE Standard for Software Unit Testing," no., 1987.*

[12] *IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.*

[13] *Aditya P. Mathur "Foundation of Software Testing" ,First Edition ,Pearson Education,2007.*

[14] *Thomas J. McCabe "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, pp. 308-320,1976.*

[15] *Zhang Zhonglin and Mei Lingxia, "An Improved Method of Acquiring Basis Path for Software Testing," 5th International Conference on Computer Science and Education, ICCSE 2010, pp.1891-1894, 2010.*

[16] *Du Qingfeng and Li Na, "White box test basic path algorithm," Computer Engineering, vol. 35, pp. 100–102,123, Augest 2009.*

[17] *Mohan V and Mala Jeya ,"intelligent ester –Test Sequence Optimization framework using Multi-Agents", JOURNAL OF COMPUTERS, VOL. 3, NO. 6, Academy Publishers,2008.*