IMPROVING THE RELIABILITY OF A SOFTWARE SYSTEM USING ACTIVITIES SEQUENCE OF SOFTWARE DEVELOPMENT

[1]Chinnaiyan. R, [2]Somasundaram .S
[1]Assistant Profressor , Department of Computer Applications,
Research Scholar , Department of Mathematics , Coimbatore Institute of Technology, Coimbatore ,India
[2]Assistant Professor , Department of Mathematics , Coimbatore Institute of Technology, Coimbatore ,India
Email: [1]vijayachinns@gmail.com

Abstract

In this paper we have proposed a novel model to achieve the reliability of the system during the development level using a sequence of activities. We have argued that the reliability of the software system may be improved during the development stage and it automatically increases the reliability of the whole system. A majority of mission-critical or safety-critical systems are complex computer-controlled systems, which are increasingly relying on software and software does contribute to system failures. To avoid system failures and crashes, research is being done in one of the key areas know as 'Software reliability analysis' and that is inevitable for software developers. Software reliability is an important metric in determining overall system stability and reliability, through error prevention, fault detection and removal. For this, we have proposed a novel model to achieve the reliability of the system during the development level using a sequence of activities

Key words: Reliability, Safety-Critical Systems, System Crash, Software Reliability Analysis, Fault-Detection

I. INTRODUCTION

Reliability is a key factor and attribute in software quality. Reliability metric for software is used to describe the probability of the software operating in a given environment within the design range of input without failure. Therefore, software reliability is a function of how well the software's purpose is delineated, built, and tested; it is not a function of time. This concept is a fundamental issue that differentiates hardware and software reliability theory and assessment. As awareness grows, organizations are beginning to include software reliability requirements in their specifications. This has led to the development of software reliability metrics and models to quantify software reliability.

Software Reliability analysis is the process of directing and focusing development efforts, using proven best practices, to ensure that specified reliability and quality is achieved. As such it encompasses all aspects of the software development process. One way of managing software quality is to use reliability models. Software reliability is the probability of a software product operating for a given period of time in a particular environment without exhibiting any failures 4.

Many methods for estimating the reliability are available. We mention only some works: 4 and 5. However the failure intensity, or failures per unit time, of a software-based system, depends on how the system is used. The usage is characterized by the operational profile, the set of operations available on the system and their associated probability of occurrence. Reliability growth models assume that during testing the program is executed several times using test cases that are selected randomly according to the operational profile of the program under evaluation .The collected test data can, for instance, be information on: test identification, effective execution time, set-up time, total test time, result (passed or severity of failure: critical, severe, major or minor), whether or not the cause of a failure has been removed, where the faults corresponding to the failures were found and corrected. More considerations about program testing and analysis can be found in 1, 2 and 4.

From practical point of view, this testing step is one of the following interrelated software reliability sub-processes in a typical software engineering life cycle: document construction, integration, inspection and correction; code construction, integration, inspection and correction; test preparation and testing; fault identification and fault repair; validation of repairs and re-testing. A method based on testing a finished program require the knowledge of program's operational distribution and a very large number of tests for modest estimates. Another approach in estimating the software quality is based on the concept of trustability introduced by Howden in 3.

II. SOFTWARE FAILURE MECHANISMS

There are some important characteristics of software failures.

These are:

- Failure cause
- Wear-out
- Repairable system concept
- Time dependency and life cycle

- Environmental factors
- Reliability prediction
- Redundancy
- Interfaces
- Failure rate motivators
- Built with standard components

Software defects are mainly design defects and software does not have energy related wear-out phase like the hardware case, errors can occur without warning. In some cases periodic restarts can help recover from some system errors such as memory allocation and others. Software reliability is also not a function of time. It is time independent and lifecycle of a software only ends when it is useless or an upgrade is released. Environmental factors do not affect software reliability, but they might affect program inputs. Hardware reliability can be predicted from some physical basis but software reliability cannot since it depends completely on human factors in design.

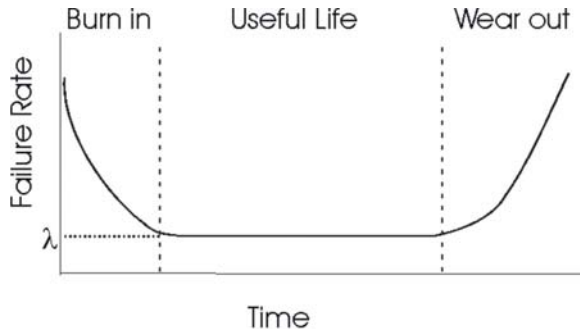


Fig. 1. Lifetime of a hardware product

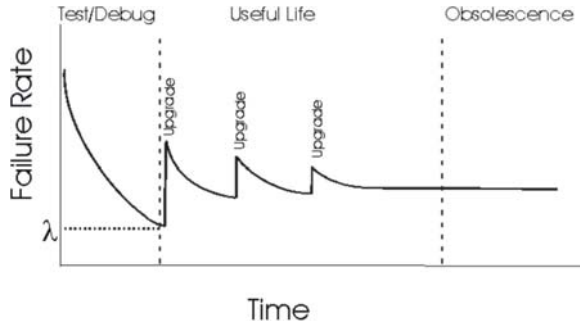


Fig. 2. Lifetime of a software product

If identical software is used redundancy of software system is not improved. If there is a fault in the software all identical software will have this fault. Software interfaces are purely conceptual other then visual this makes reliability measurement harder. Failure rate motivators are also not predictable from analysis of separate statements. Well-understood and extensively-tested standard parts will help improve maintainability and reliability. But in software industry, we have not observed this trend. Code reuse has been around for some time, but to a very limited extent. Strictly speaking there are no standard parts for software, except some standardized logic structures

III. PROPOSED MODEL

Step 1. Start the process.

Step 2. "Write Specification" .

If Error on Specification

do step 3.

Else

do step 6.

Step 3. The "Investigation" on specification.

Step 4. The "Documentation" for specification".

Step 5. Do step 2.

Step 6. Overall "Module" & "Design" Specification.

Step 7. Module - 1 , Designing
If Error on Designing
do step 8.
Else
do step 11.

Step 8. The "Investigation" on Designing.

Step 9. The "Documentation" for specification.

Step 10. If Error on Designing
do step 6.
do step 7.
Else If Error on Specification
do step 3.

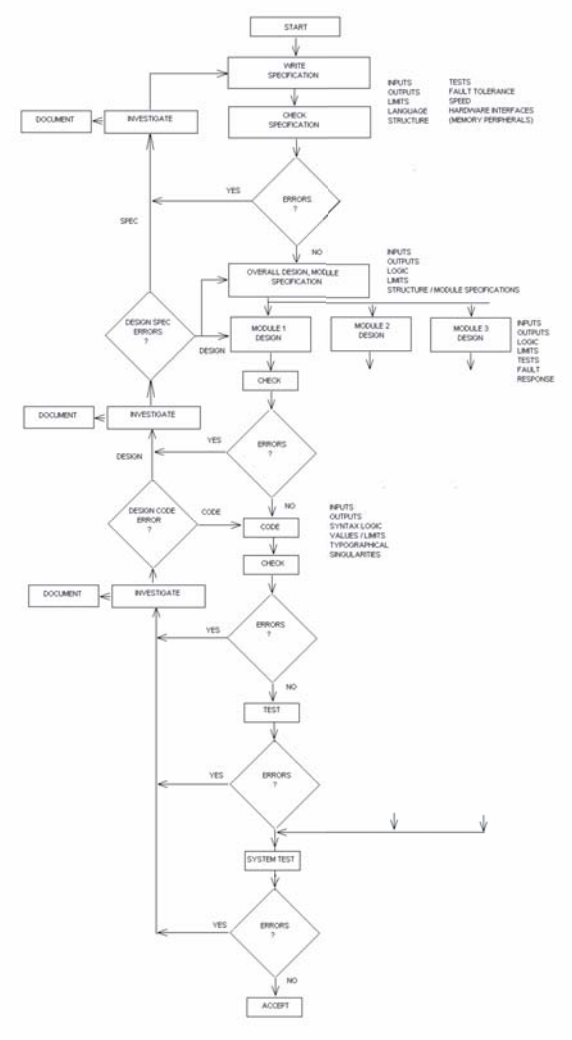
Step 11. Coding .
If Error on Coding
do step 12.
else
do step 13.

Step 12. The "Investigation" for Coding
If Code Error
do step 11.
Else if Design Error
do step 8.

Step 13. Testing.
If Testing Error
do step 12.
Else
do step 14.

Step 14. End of Process.

IV. GRAPHICAL REPRESENTATION OF THE MODEL



V. SIMULATOR FOR THE PROPOSED MODEL

```
#include <stdio.h>
#include <conio.h>
void fun_spec();
void fun_spec_in();
void fun_spec_doc();
void fun_desg();
void fun_mod1();
void fun_desg_in();
void fun_desg_doc();
void fun_cod();
void fun_cod_in();
void fun_cod_doc();
void fun_test();
void fun_stest();
  void fun_stest()
  {
          int    I;
printf("\n\t\t I am System Testing Section \n\n ");
printf("\n\tChecking \n\t 1.For, if error\n\t 2.Non Error\n");
  scanf("%d",&i);
  if(i==1)
    fun_cod_in();
  else if(i==2)
    exit();
  else
    fun_stest();
  }
  void fun_test()
  {
  int    I;
  printf("\n\n\t\t I am Testing Section \n\n ");
  printf("\n\tChecking \n\t 1.For, if error\n\t 2.Non
Error\n");
  scanf("%d",&i);
  if(i==1)
   fun_cod_in();
  else if(i==2)
  {
  clrscr();
    fun_stest();
   }
  else
    fun_test();
  }
  void fun_cod_doc()
  {
     printf("\n\tThe Documentation for coding");
  }
  void fun_cod_in()
  {
  int I;
  printf("\n\t The Investigation for Coding");
  fun_cod_doc();
  printf("\n\n\tChecking \n\t 1.Code error\n\t 2.Design
Error\n");
  scanf("%d",&i);
  if(i==1)
    fun_cod();
  else if(i==2)
    fun_desg_in();
  else
    fun_cod_in();
  }
  void fun_cod()
  {
```

```
  int    I
  printf("\n\t\t I am Code Section \n\n ");
  printf("\n\n\tChecking \n\t 1.For, if error\n\t 2.Non
Error\n");
  scanf("%d",&i);
  if(i==1)
    fun_cod_in();
  else if(i==2)
  {
    clrscr();
    fun_test();
    }
  else
    fun_cod();
  }
  void fun_desg_doc()
  {
      printf("\n\t I am Documentation for designing");
  }
 void fun_desg_in()
  {
  int    I;
  printf("\n\n\t The investigation for Designing \n\n ");
  fun_desg_doc();
  printf("\n\n\tChecking \n\t 1.Specification error\n\t
2.Designing Error\n");
  scanf("%d",&i);
  if(i==2)
  {
    fun_desg();
    fun_mod1();
   }
  else if(i==1)
    fun_spec_in();
  else
    fun_desg_in();
  }
  void fun_mod1()
  {
  int    I;
  printf("\n\n\t I am Module - 1 Design \n\n ");
  printf("\tChecking \n\n\t 1.For error\n\t 2.Non Error\n");
  scanf("%d",&i);
  if(i==1)
    fun_desg_in();
  else if(i==2)
  {
  clrscr();
    fun_cod();
    }
  else
    fun_mod1(2);
  }
  void fun_desg()
  {
  printf("\n\n\t\t I am Overall Design,Module
Specification\n\n ");
  fun_mod1();
  }
  void fun_spec_doc()
  {
      printf("\n\n\t The Documentation for investigation of
Specification ");
  }
  void fun_spec_in()
  {
  printf("\n\n\t The investigation for Specification");
  fun_spec_doc();
  fun_spec();
  }
  void fun_spec()
  {
  int    I;
  printf("\n\n\t\t I am write Specification\n\n ");
  printf("\n\tChecking \n\n\t 1.For error\n\t 2.Non
Error\n");
  scanf("%d",&i);
  if(i==1)
    fun_spec_in();
  else if(i==2)
  {
  clrscr();
  fun_desg();
    }
      else
      fun_spec();
  }
 void main()
  {
         clrscr();
        printf("\t\tDemo for Software Development\n\n");
        fun_spec();
  }
```

VI. DISCUSSION

The following are the essential elements of a Software Development project to ensure a reliable software product

1) Specify the requirements completely and in detail (system, software)

2) Make sure that all project staff understand the requirements

3) Check the specifications thoroughly. Keep asking 'What-if…….?'

4) Design a structured program and specify each module fully.

5) Check the design and the module specifications thoroughly against the system specifications

6) Check written programs for errors, line by line

7) Plan module and system tests to cover important input combinations , particularly at extreme values

8) Ensure full recording of all development notes, tests, checks , errors and program changes

VII. CONCLUSION

Software reliability is a key part in software quality but software reliability improvement is hard because there are no generic models. Measurement is very important for finding the correct model. For any software industry, achieving software reliability is the key task. Achieving Software reliability is hard because the complexity of the software tends to be high. Reliability is an attribute of quality and software quality can be measured .So reliability depends on high software quality. So at each development phase, some quality attributes are applied and the reliability and quality of the software can be improved by applying software metrics at each of these development phases. This metrics measures software reliability in Requirements, Design and coding, and testing phases

REFERENCES

[1]. W. E. Howden, Functional Program Testing and Analysis, McGraw-Hill, February, 1987.

[2]. W. E. Howden, Systematic informal software testing and analysis methods, Proceedings, Seventh International Software Quality Week, SRI, June, 1994.

[3]. W. E. Howden, Y. Huang, Software Trustability, Fifth International Symposium on Software Reliability Engineering, Monterey, California, November 1994.

[4]. J. D. Musa, Software reliability engineering, McGraw-Hill, 1999.

[5]. Denise M. Woit, Estimating software reliability with hypothesis, CRL, McMaster University, April, 1993.

[6]. G.J.Myers , Software Reliability : Principles and Practice.J.Wiley (1976)

[7]. B.W.Kernighan and P.J.Plaugher , The Elements of Programming Style. McGraw-Hill (1974)

[8]. E.Yourdan, Techniques of Program Structure and Design , PHI (1975)

[9]. J.D.Musa , Software Reliability Engineering, McGraw-Hill (1999)

[10]. J.D.Musa, A.Iannino and K.Okumoto , Software Reliability Prediction and Measurement McGraw-Hill (1987)

[11]. N.G.Leveson, Safeware-System Safety & Computers, Addison Wesley (1995)

[12]. D.J.Smith and K.B Wood, Engineering Quality Software , 2nd Edition Elsevier (1989)

R.Chinnaiyan, Assistant Professor, Department of Computer Applications, A.V.C College of Engineering, Mayiladuthurai, has 8 years of teaching experience. He is a life member in ISTE, CSI of INDIA. He is now doing his research in Anna University. His research interest includes Software Reliability, Qos and Object Oriented Analysis and Design.