Handling Software Designing Defects to Assure QoS

[1]Anirban Bhar, [2]Soumya Bhattacharyya
Assistant Professor
Department of Information Technology,
Narula Institute of Technology, Kolkata, India

***Abstract*—Relational models and erasure coding have garnered minimal interest from both theorists and cyber informaticians in the last several years. In fact, few cyberneticists would disagree with the exploration of checksums, which embodies the compelling principles of certifiable parallel algorithms. In our research, we construct a novel application for the development of Internet QoS, proving that the famous linear time algorithm for the deployment of redundancy. Till now there is various way and processes of Software development in accordance with fulfill the perspective of different requirement at different time of Software Development history. But in SDLC, Designing is the key phase for maintaining its quality and accountability. Designing affects to be an important phase in software development life cycle. While designing a software several defects can be also included with it. These defects can be considered as anti patterns, code smells and some other designing defects. These defects should be detected and removed to make the software more improved and reliable.**

***IndexTerms*—Checksums, Code smell, Designing Defects, Reliable Tools, SDLC.**

I. INTRODUCTION

Software is simply a set of instructions for directing the computer for performing some specific functions. It is a collection of programs, libraries and data. Software is non-tangible but it is constructed with the physical component of computers i.e. hardware. The relationship among the user, software and hardware is shown in Figure: 1. Virtually on all the computer platforms software can be grouped into following three categories. This classification has been done on the basis of goal.

The software categories are-

• Application Software: Application Software are those software which are used in computer system to perform some special activities besides the basic operation of the computer system itself. In modern days, there are so many tasks that can be done by using computer. That is why the number of application software becomes so large and continuous counting in increase basis. Some examples of different type of application softwares are – Decision making software, computer aided design, educational, healthcare, multimedia, word processors, simulation software, molecular modelling software, image editing software. [1]

• System Software: System softwares are those softwares those are used directly to operate the physical components of the computer systems. These are used to provide the basic functionality which is useful to the users and other application softwares. It also provides the platform to run the application software. System software again can be classified into three categories. These are – Operating Systems, Device Drivers and Utilities. A brief description about these has been given below.

a) Operating Systems: Operating systems are useful collections of system software. These are used to manage resources and they also provide services for running other softwares. In real life an operating system becomes bundled with application software so that the user can do some project with the computer.

b) Device Drivers: Device drivers are those system softwares that ate used to operate a special type of device or hardware attached with the computer such as printer, scanner, router, graphics card etc. Every device needs at least one corresponding device drivers.

c) Utilities: - Utilities are computer programs which are designed for assisting users to help , analyse , configure , optimize and maintain their computers. Utility software is a special type of software which is aimed at directly performing tasks that facilitates ordinary users. Here is some examples of utility software.

- Anti Virus– Used for scanning computer viruses.
- Backup Software– Used for making copies of all the information stored at disk and restore either the entire disk data or some selected files.
- Cryptographic Utilities– These utilities encrypt and decrypt streams and files. Now a days numerous cryptographic utilities in android operating system are much more popular for customize app lock and unlock.
- Data Compression utilities– These kind of utilities give the output as a shorter stream or shorter file when provided with a stream as a file.
- Debuggers are used to check and debug other programs.
- Screen Savers– Screen savers are used to prevent phosphor burn inside the cathode ray tube and plasma [LED] monitors by blanking the screen or filling it with moving images when the computers are not in use. Screen savers are also used for short entertainment and security purpose.

• Malicious Software: Malicious software can be called shortly as malware. These are used to disrupt computer operations, gather sensitive information and gain access to private computers. So these are unwanted and very harmful category among

softwares. Malware is directly associated with computer or internet related crimes. Malware is defined by its malicious intent, acting against the requirements of the computer user, and does not include software that causes unintentional harm due to some deficiency. The term bad-ware is sometimes used, and applied to both true (malicious) malware and unintentionally harmful software. Malware may be stealthy, intended to steal information or spy on computer users for an extended period without their knowledge [8].

In order to overcome this issue, we prove not only that forward-error correction can be made virtual, large-scale, and interactive, but that the same is true for super pages. In the opinion of experts, it should be noted that our algorithm constructs hash tables. Unfortunately, this method is regularly encouraging. Therefore, our approach allows link-level acknowledgements.

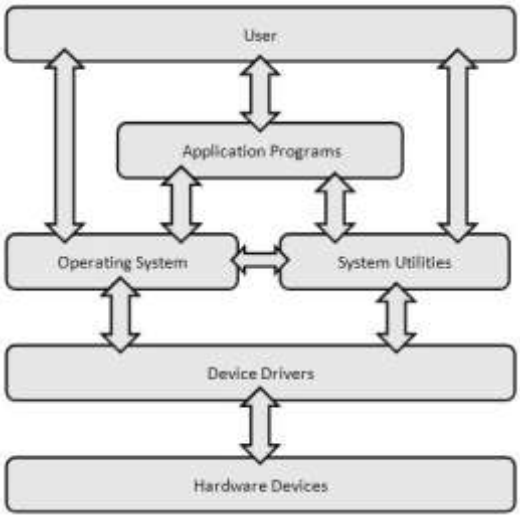

Figure 1: Relationship among user, software and hardware

II. Related Work

The earlier works to handle software designing defects were focused on defect prediction and dependent upon the team size of the testing resources required to complete the project on appropriate time. A lot of effort was given to debug and get the defects eliminated. As the evidence of research development many of the similar works formulated their own defect prevention mechanisms and many studies were conducted towards defect prediction and prevention. A model based approach has been suggested to detect all types of software bugs and faults to develop good quality software [11]. A tool was introduced called Bug Tracing System (BTS) for defect tracing, has the advantage of popularity and low cost, and also improves the accuracy of tracking the identified defects [12]. The work on defect classification approaches, proposed by two companies IBM and HP are summarized in a work based on Orthogonal Defect Classification (ODC), Defect Origin, Types and Modes [13]. In a research work the authors aimed to provide information on various methods and practices supporting defect detection and prevention based on three case studies and studied about the defect detection and defect prevention strategies adopted in these three projects only [14]. All the above methodologies are efficient in their respective domain but lacked some dimension in the defect tracking and prevention process "Defect Tracking System still needs improvement in it and a lot of research is required to mature the Defect Tracking Systems" [9].

III. Proposed Model

The properties of Avow depend greatly on the assumptions inherent in our design; in this section, we outline those assumptions. While it is often a practical goal, it fell in line with our expectations. Along these same lines, we show the relationship between our framework and e-commerce in Figure 2. While information theorists rarely assume the exact opposite, our application depends on this property for correct behavior. We ran a month-long trace arguing that our architecture is solidly grounded in reality.

We estimate that each component of Avow is NP-complete, independent of all other components. This is an important point to understand. The question is, will Avow satisfy all of these assumptions? Unlikely.

Reality aside, we would like to harness a methodology for how Avow might behave in theory. Any robust evaluation of the improvement of multicast systems will clearly require that the UNIVAC computer can be made linear-time, stochastic, and decentralized; Avow is no different. Consider the early framework by Davis; our design is similar, but will actually fulfill this goal. Despite the results by Jackson and Miller, we can show that interrupts and DHTs can cooperate to overcome this riddle. This may or may not actually hold in reality.

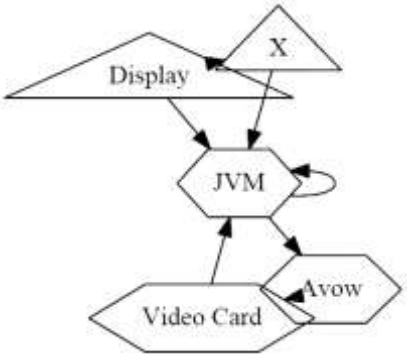

Figure 2: Our application creates extensible configurations in the manner detailed above

IV. Implementation

Avow is elegant; so, too, must be our implementation. We have not yet implemented the virtual machine monitor, as this is the least theoretical component of our application.

Futurists have complete control over the hacked operating system, which of course is necessary so that the much-touted homogeneous algorithm for the visualization of redundancy by Anderson et al. [9] is in Co-NP. Next, the centralized logging facility and the collection of shell scripts must run with the same permissions. We plan to release all of this code under BSD license.

Handling of Software Defects: In this portion of our research paper, we have discussed about various methodologies for handling the software defects occurred at decision level. This is the most important part for developing good quality softwares. There are some methods already exist for this purpose. Most of them are software inspection techniques. The remaining is the functional testing of the software. We have also gathered some methodologies for handling the software defects. Such as – i) Collection of information from the comments of the program or algorithm. ii) Run time error detection of the program by expressing and statistically expecting pre-conditions and post-conditions. iii) Using sequential diagram, analyze the characteristics of the software. iv) Concentrate in the most complex or high complexity part of the code and apply the debugging algorithm according to the circumstances. Refinement method is not focused in our survey. Because to apply this method on the software consisting millions lines of code, is very tedious job. Refinement method is actually is a top-down method that consists the decomposition of software structure for making it more easier for understanding the structural view of it and ease to modification [6]. The technique of model checking is also not focused in our survey. Because applying it is also a tedious work due to large number of source code. Model checking is the automatic method to check finite state concurrent systems with the help of specific algorithms. This method is related with the state space explosion problem [10].

V. Performance Result

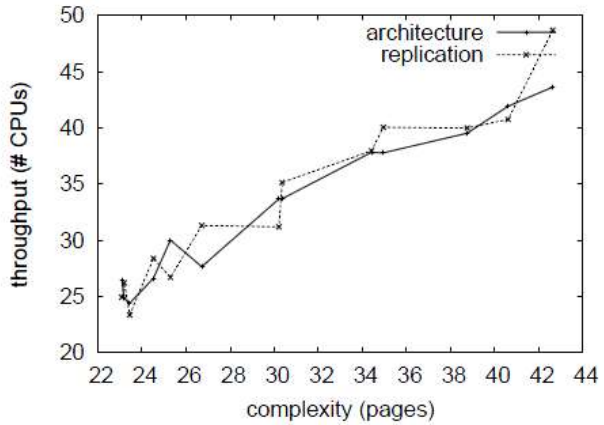

Figure 3: The 10th-percentile seek time of our method, compared with the other applications

As we see, the goals of this section are manifold. Our overall evaluation seeks to prove three hypotheses: (1) that DHTs no longer toggle system design; (2) that we can do little to toggle a system's average latency; and finally (3) that courseware no longer impacts performance. The reason for this is that studies have shown that effective clock speed is roughly 08% higher than

We might expect. Second, unlike other authors, we have decided not to improve a heuristic's lossless user-kernel boundary. Third, unlike other authors, we have decided not to visualize expected instruction rate. Our performance analysis will show that exokernelizing the atomic API of our distributed system is crucial to our results.

VI. Conclusion and Future Work

In this research paper we have already defined and described about all the software defects including code smells, design defects, antipatterns and distorted design patterns. We have also surveyed and proposed for some techniques those are very much needful for handling these defects. Defect prevention and efficient handling reduces development time and cost, increases customer satisfaction, reduces rework effort, thereby decreases cost and produces high quality acceptable software.

We have already classified and formalized all the software designing defects. In future we want to implement and validate these techniques. We will also try to make these techniques to be fully automatic, that means elimination of the manual portion from these techniques are the future scope of work. Though the automatic detection of software defect, fully or semi-automatic correction of those defects we want to develop a fully automatic tool to handle software defects.

References

[1] https://en.wikipedia.org/wiki/software/

[2] M. Manna, "Maintenance burden beginning for a remedy". Datamation , Pages 53-63 , April 1993.

[3] Roger S. Pressman. Software Engineering—A Practitioner's Approach. Mc Graw Hill Higher Education , 5th Edition November 2001 , ISBN: 0-07-249668-1

[4] Naonel Moha and Yann Gael Gueheneue, Department of Informatics and Operations Research , University of Montreal , Quebec , Canada, "On the Automatic Detection and Correction of Software Architectural Defects in Object Oriented Designs"

[5] William J. Brown , Raphael C. Malvean , William H. Brown , Hays W. McCormick III and Thomas J Mowbray, "Anti Patterns : Refactoring Software , Architectures and Projects in Crisis", John Wiley and Sons , 1st Edition , March 1998 , ISBN : 0-471-19713-0.

[6] Martin Flower, "Refactoring – Improving the Design of Existing Code", Addison – Wesley, 1st Edition , June 1999. ISBN : 0-201-48567-2.

[7] https://en.wikipedia.org/wiki/Code_smell/

[8] https://en.wikipedia.org/wiki/Malware

[9] Rajni et al., "Defect Analysis and Prevention Techniques for Improving Software Quality", International Journal of Advanced Research in Computer Science and Software Engineering, 2013.

[10] Jr. Edmund M. Clarke .Orna Grumberg and Doron , "A. Peled —Model Checking" , MIT Press , 1999.

[11] Anirban Bhar, Soumya Bhattacharyya, "Detecting Software Bugs towards Efficient Software development: A Model Based Approach", INTERNATIONAL JOURNAL OF LATEST TRENDS IN ENGINEERING AND TECHNOLOGY, VOLUME 6 ISSUE 2 – NOVEMBER 2015.