Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 5, Issue. 2, February 2016, pg.01 – 06

An Approach to Handle Software Designing Defects to Produce High Quality Acceptable Software

Anirban Bhar[1], Soumya Bhattacharyya[2]

[1]Assistant Professor, Department of Information Technology, Narula Institute of Technology, India
anirban.bhar1983@gmail.com

[2]Assistant Professor, Department of Information Technology, Narula Institute of Technology, India
sat.soumya@gmail.com

Abstract— Software Development has started its journey for betterment of human being and to build an ideal future world which surrounds with optimum needful software. Till now there are various way and processes of Software development in accordance with fulfil 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. But these suffer for lacking of tools. In software project management still there is evolution in terms of R&D done for designing the software in a very efficient and time consuming way. Several long-standing problems in software designing are concerned with inadequate requirements elicitation, analysis, specification, validation, and management. This deficit is a major cause of project failure and as such several techniques and frameworks have been developed to assist developers in handling requirements. In this paper we have discussed in brief about these defects and then an approach has also been introduced for handling those defects. We have also faced some difficulties before reaching the goal which we have vividly discussed in below.

Keywords— Antipattern, 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 another 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 badware 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]

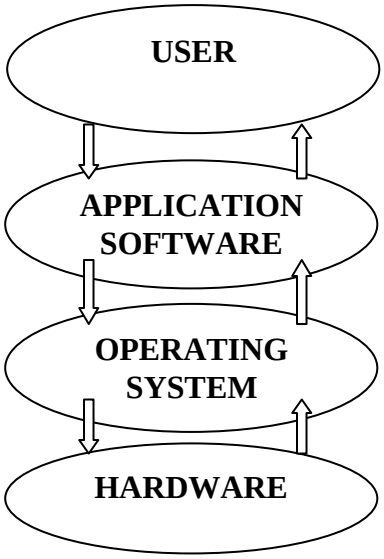

Figure 1: Relationship among user, software and hardware

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 APPROACH

In this paper we have introduced an automatic handling of software defects at designing level in terms of tools and techniques. Automatic handling of software defects is very essential to ease the maintenance and also to reduce the maintenance effort of the software.

However maintenance of existing software represents over 60% of all effort expended by a development team [2]. Only about 20% of all maintenance effort is expended to fix defects whereas the remaining 80% effort is expended to adopt the change for existence of their changing external environment , making enhancement on the basis of user request and reengineering or sometime reverse engineering to build application for future use [3].

However maintaining a software is a tedious work and sometimes it becomes impossible due to huge amount of code lines. Reengineering is one of the best solutions to facilitate and to improve software maintenance. Here we have proposed for detection and correction of software defects at the very designing level. In the first part we have discussed about some software defects aroused at designing level. Then in the next part we have proposed the detail of detection and correction of software defects in terms of tools and techniques. We have also concluded with some challenges that we have faced during research period.

Categories of Software Defects: [Figure: 2]A transparent description of different categories of software defects and their classification is very needful before describing any techniques for handling those defects

unless the debugging method or technique is not as much as fruitful which is expected at per. A brief description about these defects is given below.

Design Defects: Design defects are similar to the modern designing patterns. Design patterns give good solutions to recursive design problems. Whereas design defects are occurring errors in the design of the software that came from the absence of design patterns or the bad implementation of designing patterns. So design defects are the distorted or imprecise forms of design patterns [4].

Anti-patterns: An anti-pattern (or antipattern) is a common response to a recurring problem that is usually ineffective and risks being highly counterproductive. A commonly used process, structure or pattern of action that despite initially appearing to be an appropriate and effective response to a problem, typically has more bad consequences than beneficial results, and A good alternative solution exists that is documented, repeatable and proven to be effective. The idea of antipatterns is to show what not to do. Antipatterns describe a commonly occurring solution to a design problem, solution which generates decidedly negative consequences. Some famous antipatterns are The Blob, The Spaghetti, The Poltergeist, cargo cult programming, death march, groupthink and the Lava Flow. The Blob corresponds to one single complex controller class that monopolizes the processing technique and is surrounded by simple data classes. The spaghetti code is one of the most famous antipattern. It describes a program or a system with a software structure that lacks clarity in designing level. It is hard to maintain because it's code is more complex [5].

Code Smells: Code smell, also known as bad smell, in computer programming code, refers to any symptom in the source code of a program that possibly indicates a deeper problem. It refers to the symptom of a problem at the code level. According to Beck and Fowler code smell can be defined as "Certain Structures in Code that suggest the possibility of refactoring." [6] Code smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality". Code smells are usually not bugs—they are not technically incorrect and do not currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future. Bad code smells are an important reason for technical debt. Most bad smells affecting a piece of code are already present since its creation, rather than being introduced later via evolutionary code changes while most code smells are introduced while adding new features and enhancing existing ones, refactoring activities can also add bad smells. [7] Examples of some famous code smells are Duplicate Code, Long Method, Long Parameter List, Large Class, Data Class etc.

Classification: All the software defects can be classified into two classes. These are intra class defects and inter class defects. Code smells are intra class defects i.e. these are related with the inner functioning of the classes. Design defects include the relationships among classes. So unlike code smells, design defects are inter class defects. Antipatterns are both i.e. these are intra class defects as well as inter class defects. So it is much more efficient to compare with the proper software designing techniques in SDLC designing phase for producing high efficient softwares.

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, analyse 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]

For correcting any kind of defects we must develop a refactoring technique. It should be generalized. So, without any modification we are capable to add new software defects. Though the detection method is an automatic method; but the correction method is semi automatic method. Detection of software defects will be done but it will depend on the maintainers of the refactoring technique, not on the content of code.
Challenges: The primary goal of our research is to handle all the software defects including code smells, antipatterns, design defects and other software defects. The defect handling technique should be generic and it should work on any type of software defects. But this handling technique is not fully automatic. Here we detect the software defects manually at first. Then we will try to check whether they are detected or not through our technique. After that a comparative study gives the deviation in the both way values. If the result is tense to zero then we can say that the technique is optimum for defect detection.

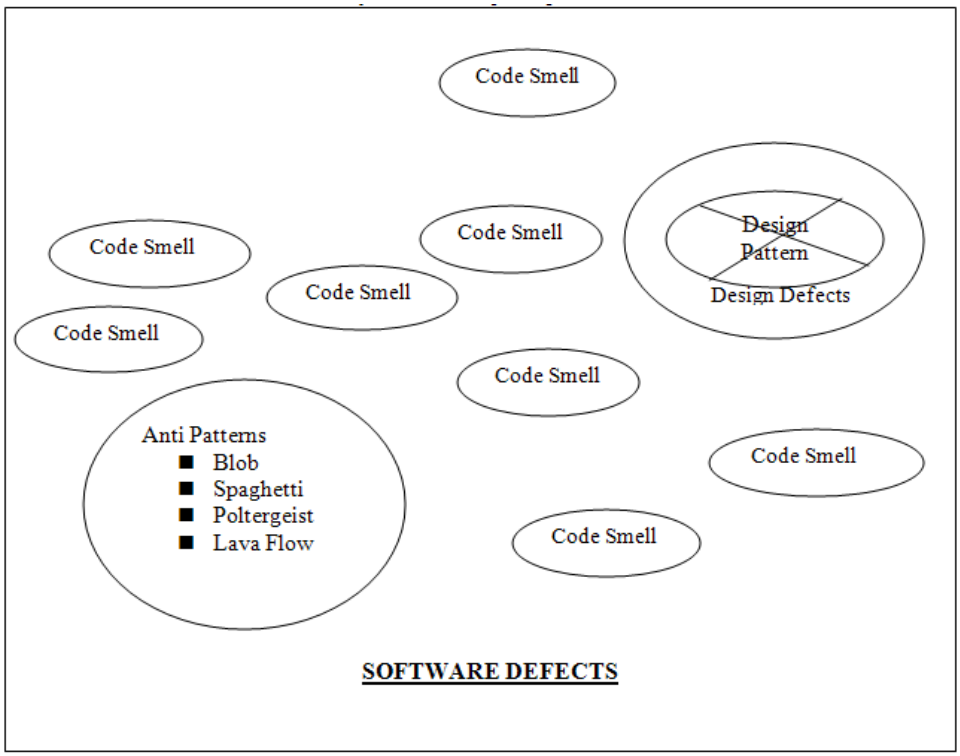

Figure 2: Categories of Software Defects

IV. 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.
[12] Pan Tiejun, Zheng Leina, Fang Chengbin, 2008, "Defect Tracing System Based on Orthogonal Defect Classification" published in Computer Engineering and Applications, vol 43, PP 9-10, May 2008.
[13] Stefan Wagner, 2008,"Defect Classification and Defect Type Revisited" Proceedings of the 2008 workshop on Defects in large software systems, (DEFECTS'08) pages 73-83, ACM Press, 2008.
[14] Suma V and T R Gopalakrishnan Nair , 2008, " Effective Defect Prevention Approach in Software Process for Achieving Better Quality Levels" Proceedings of World Academy of Science, Engineering and Technology Volume 32 August 2008.

