FINE–GRAINED ANALYSIS AND PROFILING OF SOFTWARE BUGS TO FACILITATE WASTE IDENTIFICATION AND ITS MINIMIZATION

S.R. Subramanya[1]

[1]*School of Engineering and Computing, National University, San Diego, CA 92123, USA*

Abstract

Software defects or bugs are among the primary causes of software development overrunning time schedules and budget costs. They are also the major cause of 'waste' in software development, which roughly translates to time, effort, and money spent on unproductive aspects of software. Despite several developments in software engineering and improvements in the software development process, their effects in minimizing the waste in software development process have not been remarkable in comparison with the hardware counterpart of complex chip design. This paper proposes a fine–grained approach to the analysis of the root causes of software defects (bugs) in an effort to better quantify the components of waste and its subsequent minimization. It also proposes the use of bugs profile for the allocation of resources to tackle bugs with minimal wasted resources.

Keywords: *Software bugs, Fine–grained analysis, Waste in software, Waste identification, Waste minimization*

----------***----------

1. INTRODUCTION

The development and maintenance of any software of reasonable complexity is necessarily a human–intensive, time consuming, and expensive process. Faults (bugs) are introduced into the software system in a variety of ways. Despite several developments in software engineering and improvements in the software development process, their effects in minimizing the faults (bugs) in software and thereby improving the reliability has not been remarkable in comparison with the design and development of complex microprocessor chips, and their reliability.

The complete avoidance of bugs may not be possible for real–world software systems, primarily due to the fact that software development is a complex, human-intensive process. However, methodologies which would minimize the introduction of bugs into the source code in the first place (as much as possible) would be highly beneficial for reducing the high software development and maintenance costs, and for increasing the quality of software.

Numerous studies have been done in effective software testing to enable the detection of most (if not all) of the bugs. For example, 28 best practices that contribute to improved software testing are listed in [3]. Numerous efforts have been put into finding methods for preventing developers from inadvertently introducing bugs [4, 5]. Several studies have been done to predict occurrences of bugs. For example, a methodology using software bug history data to model and predict future bug occurrences is presented in [10]. In addition to code reviews, proactively improving code quality using static and dynamic analysis is given in [2]. Analysis of some of the root causes of bugs along different dimensions such as (a) management–related, (b) design–related, (c) programming–related, and (d) human–factors–related, is given in [9]. Results from various studies have been compiled into an extremely useful and interesting list of ten items containing statistics and causes of several kinds of software defects, and means of their reduction, and is presented in [1].

It is now a well–known fact that when defects are found later in the development lifecycle, they are going to take (exponentially) more time and cost more money to fix them than if they were discovered sooner. Since software bugs are the primary 'components' of waste, it would thus be beneficial to identify and eliminate (or at least minimize) bugs early in the process, thereby minimizing waste in the overall development process. In order to do this, this paper presents a methodology of 'fine-grained' analysis of the causes of bugs, leading to the fine, measurable granular–causes which make up the causes. These granular-causes are better understood and steps can then be devised to tackle them. In addition, using the Pareto principle, the bugs can be analyzed, and the allocation of resources can be optimized to tackle the bugs.

The next section gives brief background on the development of waste in software. Section 3 presents the proposed fine–grained analysis of the factors contributing to software bugs to derive granular causes and their use in expressing the causes of bugs. Section 4 describes the use of profiling to determine the hot spot modules contributing to bugs and a case of putting resources for tackling bugs in them, which is followed by conclusions.

2. BACKGROUND

The notion of waste in manufacturing was popularized by the Toyota Production System. Since then numerous studies have been done to adapt the notions of waste and their elimination/minimization in the domain of software. The notion of waste in software development can be traced to [7] where the term lean software development was introduced. [8] gives a translation of the seven identified wastes in a manufacturing system into the seven wastes of Software Development, namely: Partially Done Work, Extra Features, Relearning, Handoffs, Delays, Task Switching, and Defects. Examples of waste in software, motivators for waste reduction, counter measures to development waste are presented in [6]. A common underlying theme contributing to waste in software is that of faults/defects/bugs. The effective avoidance, prediction, detection, and correction of bugs have been elusive, and have been the subject of numerous studies.

3. FINE-GRAINED ANALYSIS OF FACTORS OF BUGS

In the proposed scheme of fine-grained analysis of bugs, first, the major factors in orthogonal dimensions which cause bugs are determined. Then, each of these factors is analyzed in detail to determine numerous issues – the granular causes – which contribute to the given factor. Each of the granular causes should be simple enough for amenable solution(s). Since each of the granular causes for a given factor may not be independent, we need to find their interdependencies and find their collective effect on the factor causing the bug.

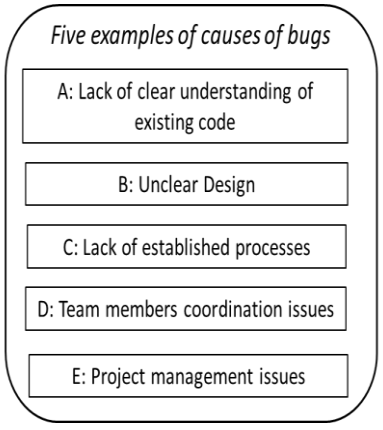

Fig -1: Examples of causes of bugs

First, as an example, we consider five major dimensions responsible for the introduction of bugs (faults) (see Figure 1), namely, (A) lack of clear understanding of existing code; (B) unclear design; (C) lack of established processes; (D) team members' coordination issues; (E) project management issues.

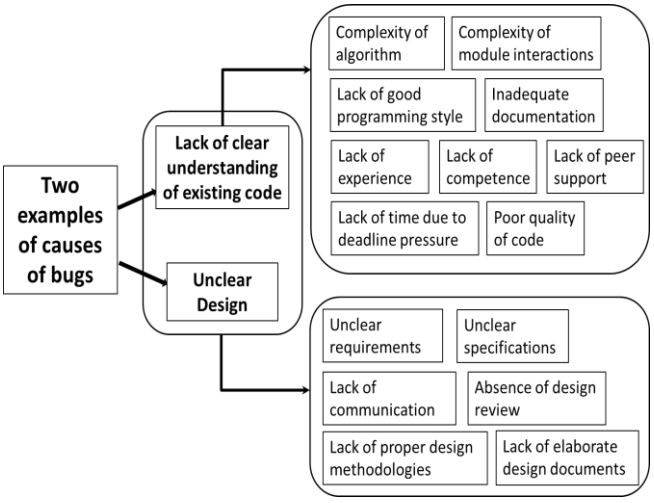

Fig -2: Examples of two causes of bugs and their composition in terms of granular causes

Figure 2 shows two of the example causes of bugs and the granular causes for each of them. This is also shown in Table 1. For example the 'unclear design' aspect contributing to the bugs has, in turn, six granular causes namely, (a) unclear requirements, (b) unclear specifications, (c) lack of communications, (d) absence of design review, (e) lack of proper design methodologies, and (f) lack of proper design documents. This is shown in Table 1.

Table -1: Two example factors and corresponding granular causes related to software bugs

Factors	Granular causes
Lack of understanding of existing code	Complexity of algorithm
	Complexity of interactions
	Lack of good programming style
	Inadequate documentation
	Poor quality of code
	Lack of experience
	Lack of competence
	Lack of peer support
	Lack of time
Unclear design	Unclear requirements
	Unclear specifications
	Lack of communications
	Absence of design review
	Lack of design methodologies
	Lack of elaborate design documents

We will now present the relationship of content consumption experience parameters with the other parameters of the factors influencing the content consumption experience. The causes of bugs, X is given by:

$$X = AA'U + BB'D + CC'P + DD'T + EE'M ,$$

where *U, D, P, T,* and *M* are vectors of granular causes corresponding respectively to the major example factors of causes of bugs, namely, *lack of clear understanding of existing code (U); unclear design (D); lack of established processes (P); team members' coordination issues (T); project management issues (M).*

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{Nx} \end{bmatrix}$$

is a vector of parameters corresponding to the causes of bugs. These parameters are as non–overlapping (orthogonal) as possible.

$$U = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{Ni} \end{bmatrix}$$

is a vector of granular causes related to the "lack of clear understanding of existing code" factor of bugs causes. The granular causes, may in turn, consist of a set of attributes, each of which will have a defined range of values. The other vectors *D, P, T,* and *M* are similarly defined.

$$A = \begin{bmatrix} a_{1,1}a_{1,2}\cdots a_{1,Ni} \\ a_{2,1}a_{2,2}\cdots a_{2,Ni} \\ \vdots \\ a_{Nx,1}a_{Nx,2}\cdots a_{Nx,Ni} \end{bmatrix}$$

is an $N_x \times N_i$ cross correlation matrix whose elements capture the dependences among the granular causes of *U* and the parameters of *X*.

$$A' = \begin{bmatrix} a'_{1,1}\, a'_{1,2}\cdots a'_{1,Ni} \\ a'_{2,1}\, a'_{2,2}\cdots a'_{2,Ni} \\ \vdots \\ a'_{Ni,1}\, a'_{Ni,2}\cdots a'_{Ni,Ni} \end{bmatrix}$$

is an $N_i \times N_i$ matrix whose elements represent the correlation among the granular causes of *U*.

Therefore,

$$AA'U = \begin{bmatrix} a_{1,1}a_{1,2}\cdots a_{1,Ni} \\ a_{2,1}a_{2,2}\cdots a_{2,Ni} \\ \vdots \\ a_{Nx,1}a_{Nx,2}\cdots a_{Nx,Ni} \end{bmatrix} \begin{bmatrix} a_{1,1}a_{1,2}\cdots a_{1,Ni} \\ a_{2,1}a_{2,2}\cdots a_{2,Ni} \\ \vdots \\ a_{Ni,1}a_{Ni,2}\cdots a_{Ni,Ni} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_{Ni} \end{bmatrix}$$

which yields a $N_x \times 1$ vector which captures the dependences among the granular causes of *U* (lack of clear understanding of existing code) as well as their dependence on the parameters of *X* (bugs causes). The other products, *BB'D*, *CC'P*, *DD'T*, and *EE'M* are similarly defined. The sum of all these products thus represents the effects of the granular causes of lack of clear understanding of existing code, unclear design, lack of established processes, team members' coordination issues, and project management issues, upon the causes of bugs.

Thus, in essence, the proposed scheme expresses the causes of bugs in terms a few (orthogonal) parameters. The parameters are expressed in terms of several factors of bugs causes, and each of the factors is expressed in terms of granular causes, each of which is simple and measurable. This facilitates the understanding of the complex relationships among the granular causes and their combined effect on the causes of bugs. This can be used to device methods to minimize waste in terms of time and effort in detecting and correcting bugs, as well as in proactively having measures to minimize (avoid) introduction of bugs in the first place.

4. PROFILING OF BUGS

In this section, we present the profiling of bugs so that the distribution of bugs across different modules in huge software can be determined, and also predicted. This enables minimization of waste by optimal allocation of resources to proactively and effectively tackle the bugs.

The *Pareto principle*, named after Italian economist Vilfredo Pareto, (also known as the 80-20 rule or the *law of the vital few*) states that, for many phenomena, 80% of the consequences stem from 20% of the causes. For example, 80% of income goes to 20% of the population, 80% of the sales come from 20% of the products, 80% of the resources are typically used by 20% of the operations, we wear our 20% most favored clothes about 80% of the time, etc.

In software engineering, it is also often the case that 80% of the development effort is spent in 20% of the system (modules), 80% of the execution time of a computer program is spent executing 20% of the code, 80% of the debugging time/effort is taken by 20% of the bugs, etc. Thus, it is important to identify the 'critical' 20% parts – the *hotspots*,

which need the most attention in terms of improvements. Improvements to this critical 20% of the software system (or process) would result in improvements in the 80% of the result that this system influences.

For example, the modules which account for the most faults/bugs can be identified, and made the targets for improvements.

4.1 Fault/bug Profiles

It is beneficial to perform 'bug profiling' – to determine which modules cause the most number of bugs. Distribution of bugs across various modules of a mobile handset software is shown in Figure 3. This is based on the actual data from the QM (quality management) group running black-box tests on the software during development. It is interesting to note that the distribution follows the ***Pareto principle*** – about 20% of the modules account for about 80% of the bugs. Of course, the number of bugs that are caused by a given module depend upon a complex set of factors including (i) how complex a module is, (ii) how clear the specifications are, (iii) how many persons are involved in the development of the module and their experience, (iv) the number of other modules that this module interacts with, etc. The important thing to be learned is that of predicting the bugs that a module could cause and taking appropriate actions proactively. For example, assigning experienced engineers, allocation of more resources as necessary, spending more effort in better design, etc., would help in minimizing the bugs, and hence the time and effort wasted.

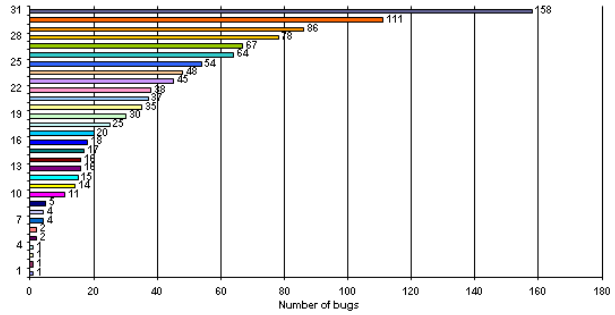

Fig -1: Distribution of bugs reported by QM for a mobile handset software during development

In the long term, it is also beneficial to study correlations between bugs and other factors such as the base lines used, number of newer features implemented, number of files touched, number of deliveries, etc. These facilitate bug predictions, and appropriate proactive solutions. Another experiment of interest is to study the spread of bugs among modules / functions, i.e., a new bug arising in a module due to a change or new code in a module results in a previously unknown bug appearing in another module. Detailed analysis of the results could be used beneficially in the design of modules with less coupling.

5. CONCLUSIONS

Software defects or bugs are the major causes of 'waste' in software development translating to time, effort, and money spent on unproductive aspects of software. This paper proposed a fine–grained approach to the analysis of the root causes of software defects (bugs) in an effort to better quantify the components of waste and its subsequent minimization. It also proposed the use of bugs profile for allocating of resources to tackle bugs using minimal resources, contributing to reduced waste.

REFERENCES

[1]. B. Boehm and V.R. Basili, "Software Defect Reduction Top 10 List", IEEE Computer, January 2001, pp135–137.

[2]. K.A. Briski, et. al., "Minimizing Code Defects to Improve Software Quality and Lower Development Costs", IBM Development Solutions Whitepaper, Oct. 2008.

[3]. R. Chillarege, "Software Testing Best Practices", IBM Research Technical Report, RC 21457, April 1999.

[4]. D. Huizinga and A. Kolawa, "Automated Defect Prevention: Best Practices in Software Management", Wiley–IEEE Computer Society Press (ISBN 0470042125).

[5]. M. McDonald, R. Musson, and R. Smith, "The Practical Guide to Defect Prevention", Microsoft Press (ISBN 0735622531).

[6]. "The Yin and Yang of Software Development: 5 Best Practices that Allow Efficiency and Creativity to Productively Coexist", Parasoft White Paper, 2013.

[7]. M. Poppendieck and T. Poppendieck, Lean Software Development: An Agile Toolkit, Addison-Wesley, 2003.

[8]. M. Poppendieck and T. Poppendieck, Implementing Lean Software Development: From Concept to Cash. Addison-Wesley, 2006

[9]. S.R. Subramanya, "Analysis of Some of the Root Causes of Bugs in a Mobile Phone Software Development Environment", International Conference on Computer Applications in Industry and Engineering, Honolulu, HI, Nov. 16–18, 2011, pp. 210–215.

[10]. C. Zhang, H. Joshi, S. Ramaswamy and C. Bayrak, "A Dynamic Approach to Software Bug Estimation", in Advances in Computer and Information Sciences and Engineering, Springer, 2008 (978-1-4020-8741-7).