IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4381

Ethical Issues in Software Engineering Revisited

Ali Salehnia, Computer Science Department, South Dakota State University, Brookings, SD 57007, Phone: 605-688-5717, E-mail: Ali_Salehnia@sdstate.edu

Hassan Pournaghshband, Department of Computer Science, Southern Polytechnic State University, 1100 South Marietta Parkway, Marietta, GA 30060-2896, Phone: 770-528-4282, E-mail: hpournag@spsu.edu

ABSTRACT

The process of software development is usually described in terms of a progression from the project planning to the final code, passing through intermediate stages such as requirement analysis, system design and coding system testing, and maintenance. One important aspect of these requirements concerns the reliability of the software. The use of computers for life-critical systems demands extremely high reliability of the computing functions as a whole. The consequences of negative results from unreliable systems and software are becoming public knowledge every day. Since these situations create a negative image for computer professionals and since these episodes create an environment of nontrust for the discipline, a good look at the ethical issues in software engineering is necessary. In this paper, we look at each of the software engineering steps and the important aspect of their reliability and safety in the analysis, design, and implementation of software. We also examine the ethical aspects of the software and system development.

1. INTRODUCTION

As software become more complex and sophisticated, so too must the methods of writing these programs. Failure to take responsibility for errors will only mean more catastrophes. The price for these failures is rising even today and it will be paid in the future by the computer professionals' through loose of dignity and trust [13]. What follows are some examples of problems raised when ethical issues in software engineering were ignored.

In a local newspaper on May 20, 1996, we found an article with the following headline: "Bank error produces 800 near-billionaires." The story was about a programming error that increased the account balance by $924.8 million dollars for each of the bank's 800 customers. This is a total of $763.9 billion, which are more than six times the bank's assets.

On May 7. 1996 in another local newspaper we found an article with the title: "Error causes 2 jets to occupy same runway." This story explains how two passenger jets came within 1,500 feet of each other on the same runway because both were assigned the same flight number.

Another article titled " Planes in Northwest lose link with air traffic control center" appeared on January 7, 1996 and the story explains how a regional center (part of a $1.4 billion computerized system) lost communications with an aircraft for a few seconds because of a software problem.

The following quotes were taken from a mug acquired at a 1982 ACM Computer Conference [2] in order to indicate to the reader how far we have come with software engineering steps and processes:

—Weinberger Law: "If builders built buildings the way programmers write programs then first woodpecker that came along would destroy civilization."

— Troutman's Programming Laws: "If a test installation functions perfectly all subsequent systems will malfunction; not until a program has been on production for at least six months will the most harmful error then be discovered; any program will expand to fill any available memory.

—Gioub's Laws of Computerdome: "The effort required to correct the error increases geometrically with time."

—Hare's Law of Large Programs: "Inside every large program is a small program struggling to get out."

The question is: What has changed or improved since 1982? Can we say that "the more thing change the more they stay the same?"

We believe that some of the problems in software development can be dealt with by computing professional if they are trained to explicitly practice ethical guidelines and accept their social responsibilities. Software developers are held responsible for the outcome of their software. Hence, they should also be held responsible if their design is at fault. Furthermore, they should assume total legal liability for their faulty and unreliable programs and they should be required to let their clients know when their systems fail to deliver something that is required of them, such as a missing function when the clients need it. Basili and Musa [15] consider such an event as a reliability problem and therefore a failure.

2. ETHICAL ISSUES AND RESPONSIBILITIES

Each of the information and computing professional organizations including the ACM, DPMA, ICCP, IEEE, etc. has a code of ethics, which emphasizes responsibility. In the general Moral Imperatives section 1.1 of the ACM Code of Ethics we read. "As an ACM member I will ... Contribute to society and human well-being.... An essential aim of computing professional is to minimize negative consequences of computing systems, including threats to health and safety. When designing or implementing systems, computing professionals must attempt to ensure that the produce of their efforts will be used in society responsible ways, will meet social needs, and will avoid harmful effects to health and welfare" [1].

The IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices (http ://www.computer.org/tab/seprof/code.htm) suggested the following as code of ethics for software engineers:

"Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles":

1. Public - Software engineers shall act consistently with the

This paper appears in *Managing Information Technology in a Global Economy*, the proceedings of the Information Resources Management Association International Conference. Copyright © 2001, Idea Group Inc.

public interest.

2. Client and Employer- Software engineers shall act in a manner that is in the best interests of their client and employer and that is consistent with the public interest.
3. Product - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. Judgment- Software engineers shall maintain integrity and independence in their professional judgment.
5. Management- Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. Profession- Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. Colleagues- Software engineers shall be fair to and supportive of their colleagues.
8. Self - Software engineers shall participate in lifelong learning regarding the practice of their profession and promote an ethical approach to the practice of the profession.

We believe that ethical issues play a big role in the analysis and development of software products. Wood, and et al. [23] argue about the need for the information systems person to receive training in ethical implications and they indicate that the existence of professional codes of practices is a clear indication that ethical neutrality is not possible. They continue to argue "Self-reflection by systems analysis on the ethical implications of their practice should ensure that ethical decisions are not made implicitly for them."

A section of the IEEE Code of Ethics is stated as follows "We the members of the IEEE do hereby commit ourselves to the highest ethical and professional conduct and agree: To accept responsibility in making engineering decision consistence with the safety, health and welfare of the public, and, to disclose promptly factors that might endanger the public or the environment. " Since software developers are considered engineers and scientists, they should definitely abide by these guidelines and produce reliable and safe products. Explaining the importance of the standards in software engineering, Lee [13] indicates, "The time has come to make software engineering a science rather than an art. Software standards must be codified and programmers must strictly adhere to those standards." Adding, "writing software programs is no less important than building a bridge, and it should be treated as such."

Balzer and Goldman propose [4,19] eight principles for good specification: Principle #1. Separate functionality from implementation; Principle #2. A process-oriented systems specification language is required; Principle #'3. A specification must encompass the system of which the software is a component; Principle #4. A specification must encompass the environment in which the system operates; Principle #5. A system specification must be a cognitive model; Principle #6. A specification must be operational; Principle #7. The system specification must be tolerant of incompleteness and augmentable; Principle#8. A specification must be localized and loosely coupled.

One problem with the traditional software development method is the specification changes during the system development. Such changes have implications that may affect all parts of the software, making the previous design inadequate [12]. A good design requires taking into account the business, personal, and social expectations of clients. Knowing the consequences of a faulty program can increase the communication and cooperation between the software design team and the users. This in turn can increase the reliability and quality of the software.

While technical expertise and know how are important in software design and development, understanding of the social systems in which the software is to be used is also very important. A majority of the computer science and information systems departments teach programming and software engineering courses only from the technical point of view rather than considering- the technical, organizational, sociological and ethical points of view. Huff and Finholt [10] argue, "A dedication to a reliable product, a commitment to open dealing with clients, and a concern for including customer and employees in the design process are pans of both the ACM ethics code and of quality design."

3. SOFTWARE QUALITY AND RELIABILITY ACHIEVEMENTS

Software reliability can be measured or estimated by using historical and developmental data [19]. Software reliability is defined as "the probability of failure free operation of a computer program in a specified environment for a specified time" [15]. A software reliability model can be used to characterize and predict behavior important to managers and software engineers. While software failure can be defined as nonconformance to software requirements. Pressman [19] believes that "all software failure can be traced to design or implementation problems. Further, he argues that "Reliability is the most costly, performance characteristic to assess and the most difficult to guarantee." It is very important to understand that the reliability of a computer program is an important element of its overall quality. If a program frequently fails to perform, it doesn't matters whether other software quality factors are acceptable or not.

Quality software is defined as: "Software that satisfies the user's explicit and implicit requirements, it well documented, meets the operating standards of the organization, and runs efficiently on the hardware for which it was developed." Software quality may be divided into three measures: operability (accuracy, efficiency, reliability, integrity, security, timely, and usability); maintainability (changeability, correctability, flexibility, and testability); and transferability (code reusability, interoperability, and portability). Ferdinand [8] states that "One can correctly opine that many modern programming practices, along with the heavy data gathering that often accompanies them, are necessary to achieve quality products, but are in themselves not sufficient for reducing- defects or significantly improving the level of software productivity sought in software engineering."

One way to ensure software quality and to achieve reliability is to use formal methods, which means the user's needs must be expressed in a mathematical language. This technique is highly reliable. However, errors can be introduced into the design during the implementation process. Another way to insure the quality and the reliability of software is to institute Software Quality Assurance functions. Dunn and Ullman [7] present a list of tasks that should be part of any software quality assurance plan: these tasks are: 1. System design; 2. Software requirements specification review; 3. Preliminary design review; 4. Detail design review: 5. Review of integration test plan; 6. code review: 7. Review of test procedure; 8. Audit of document standards: 9. Configuration control audit; 10. Test audit, 11. Define data collection, evaluation and analysis; 12. Tool certification; 13. Vendor and contractor oversight; and 14. Record keeping.

Also to ensure software quality and reliability the designers should consider and enforce software security. Describing computer security, Pfleeger [18] indicates that the attack on software

may occur by the act of modification, interruption, deletion, and interception. Changing a bit in its code can modify a program and may causes the system to crash,. The three well-known categories of software modification can be listed as: 1. Trojan Horse— a program that overtly does one thing while covertly does another; 2. Trapdoor— a secret entry point to a program and 3. Program leaks— a program making information accessible to unintended people or programs [18]. Pfleeger [18] also indicates that a program must be secure enough to exclude outside attack and must be developed and maintained so that one can be confident of the dependability of the program.

The ultimate goal of software quality is user satisfaction. Basili and Musa [5] listing important attributes that satisfy users state, "The attributes most often named as significant are functionality, reliability, cost, and product availability date. Reliability often ranks first." However, some obscure errors can have disastrous consequences. These reviews are also mandated by the Department of Defense as part of the formal requirements for a contractor's quality assurance program [12].

The third way to achieve high reliability is fault tolerance. Fault-tolerant computing is one method for increasing a system's useful lifetime. It should be not that for a given application, a system could be sufficiently reliable without fault tolerance. Furthermore, we would like to point out that using fault tolerant doesn't necessarily Guarantee that a system will be sufficiently reliable for a particular application. Availability is the probability that system will be operational at any given moment [9]. Software fault tolerance allows a system to detect errors and to avoid the failures, which result from them. If this objective is accomplished, errors cause little or no visible degradation of performance or reliability. The ultimate goal of software fault tolerance is to increase the reliability, availability, and or safety levels in critical applications.

Software fault tolerance (SFT) techniques fall into three groups: dynamic redundancy, fallback methods, and error isolation. The fault-tolerance techniques used in real-time control systems seldom conform to the basic paradigm of software fault tolerance, recovery block or N-version programming, but rather tend to be combinations of the three [3,6, 14].

Since the ultimate coal of software fault tolerance is to increase the reliability, availability, and safety of critical applications and since the ultimate goal of a CASE tool technology is to separate the software system design from the implementation of program code, therefore, the CASE tools should be very important for SFT systems development. While the classical approaches to software fault tolerant system does increase the complexity of the software systems.

Fault-tolerant strategies are concerned with keeping the software system functioning in the presence of errors. Strategies fall into three groups: dynamic redundancy, fallback method and error isolation [16]. One approach to dynamic redundancy is a processing technique known as voting. Data is processed in parallel by multiple identical devices and the output of these devices is compared. If a majority of the devices have the same result that result is assumed to be the correct one. Fallback or degraded-service methods are appropriate when it is essential for the software system to shutdown.

For instance, in a process-control system, if a software error is detested that is causing the system to fail; a separate back up piece of software may be loaded and executed to guarantee the safe shutdown of all processes being controlled by the system. The problem of common software faults given rise to similar errors in redundant software still remains open. It may be that careful analysis of a system design and requirement can identify dangerous or difficult components of the software are far special care during later phases of development and tests.

3.1. REQIREMENT ANALYSIS

A requirement analysis tool allows a system engineer to initiate, design, complete, modify, and maintain SFT systems. CASE tools can also provide special features for requirement analysis and documentation of the SFT systems. Functional requirement documents are often cast in a yen, specific, predefined format that identify each and even, requirement down to every minute detail. Each requirement in the requirements' specification list should have the features and constraints of each component along with the whole structure of the SFT systems, or/and a technique for SFT systems such as RB, N-version Programming, CBS, or RNB.

Requirement trace ability is an important method of demonstrating SFT structures produced (or reliability of the product) to satisfy user requirements. Usually, this is demonstrated in steps by showing that the reliable SFT systems produced by the current development step can be traced back to the previous step of the software life cycle. For example, we should be able to trace back to the requirement specification of SFT structure from the design specification, or from the source code to the design specification.

The requirement of SFT systems trace ability at the code-back-to-design is shown by automatically creating a structure chart from the source and comparing this chart to the structure chart created during the design phase. Therefore, the requirement specification should show the structure of SFT systems, components of the SFT approach, and constraints of the SFT systems used in the project.

3.2. SYSTEM DESIGN PHASE

Quality software is organized as a set of independent modules, each of which can be designed and tested separately. Each module views the other as black box with well-defined sets of inputs and outputs. Each module is accessed only through these inputs and outputs. This logical segregation of functionality is not only a key factor for ordinary software development but also a main factor of SFT systems implementation.

4. CONCLUSION

The process of software development is usually described in terms of a progression from the project planning to the final code, passing through intermediate stages such as requirement analysis, system design and coding system testing, and maintenance. One important aspect of these requirements concerns the reliability of the software. The use of computers for life-critical systems demands extremely high reliability of the computing functions as a whole. Applications that pace the state of art in this regard include spacecraft.

Fly-by-wire systems for passenger aircraft, safety systems for nuclear reactor, and traffic control system for tracked vehicles and aircraft. The need for high reliability of software components of these life-critical systems has become more apparent with the increasing functionality being ascribed. One result of this increased functionality is the recommendation of various designs for achieving fault tolerant system.

The ultimate goals of a software system are to increase the quality, reliability, and availability, and safety of critical applications. Even though, the classical approaches to software quality, reliability, and fault tolerant systems increase the complexity and cost of large software systems, the implication of ethical issues can address same of these problems. An engineering approach with

ethical issues in mind will enable the software developers to produce reliable software and in accordance with user requirements.

REFERENCES

"ACM's Code of Ethics and Professional Conduct". Communication of the ACM, Vol. 36, No. 12, 1993.

Art 101. Limited, Atlanta, GA. 1982

Avizienis. A. (1985). 'The N-version approach to Fault-Tolerance Software." IEEE Trans. Software Eng. No. 1 SE-11. Pp 1491-l497. Dec. 1985.

Balzer. R. and N. Goodman (1979) . "'Principle of Good Software Specification," Proc. of Specifications Software IEEE, 1979, pp. 58-67.

Basili, V. and J. Musa. (1991). "The Future Engineering of Software: A Management Perspective," ' Computer, IEEE, Vol.24. No.9, September 1991.

Cha, S. (1986). "A Recovery Block Model and its Analysis." Proc. Fifth IFAC Workshop on o safety of Computer Control Systems (SAFECOMP 1986), Oxford:Press, Pp 11-26.

Dunn, R, and R. Ullman (1982). Quality Assurance for Computer Software, McGraw-Hill.

Ferdinand, Arthur (1993). Systems Software and Quality Engineering. Van Nostrand Reinhold.

Gantenbein, r., Shin, S. and Wang, Z. (1991). "Software Fault Tolerance in a Distributed Real-Time Control System.' Proc. of 4th ISMM/IASTED International Conf. on Parallel and Distributed Computing and Systems. Washington D.C. Oct., 1991, pp 61-64.

Huff, C. and T. Finholt (1994). Social Issues in Computing: Putting Computing in its Place. McGraw-Hill.

Johnson, D. (1995). Computer Ethics. Prentice-Hall.

Johnson, D. and H. Nissenbaum (1995). Computers, Ethics, and Social Values. Prentice-Hall.

Lee, L. (1992). "Computer Out of Control". Byte, Feb. 1992, P. 344.

Leveson, N. (1987). "Building Safe Software," Software Reliability: Achievement and Assessments (B. Littlewood, editor). Oxford: Beackwell Scientific Publications, pp 1-18.

Musa, J.D.. A. Iannino, and K. Okumoto (1987). Engineering and Managing Software with Reliability Measures, McGraw-Hill, 1987.

Nelson, S. (I989). " 'Making the Business Case for CASE Technology" Mainframe Update.

Oz, Frank (1994). Ethics for the Information Age. B&E Tech.

Pfleeger, Charles (1989). Security in Computing. Prentice-Hall, 1989.

Pressman, Roger (1987). Software Engineering, McGrawHill, 1987.

Shin, S. and Alishiri. Z. (1994). "CASE Tools Comparisons". Proceedings of the Association of Management.

Shin, S., Salehnia, A. and Cong, B. (1993). "Implementation of Software Fault Tolerant Systems, " Proceedings of 1993 International IRMA Conference. Pp. 466.

Summerville, I. (1989). Software Engineering. Reading, MA: Addison-Wesley.

Wood, A.T. et al. (1996). "How We Profess: The Ethical Systems Analyst." Communication of the ACM, March 1996, Vol. 39, No. 3, pp 69-77.

0 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage:
www.igi-global.com/proceeding-paper/ethical-issues-software-engineering-revisited/31621

Related Content

Exploring Enhancement of AR-HUD Visual Interaction Design Through Application of Intelligent Algorithms
Jian Teng, Fucheng Wanand Yiquan Kong (2023). *International Journal of Information Technologies and Systems Approach (pp. 1-24).*
www.irma-international.org/article/exploring-enhancement-of-ar-hud-visual-interaction-design-through-application-of-intelligent-algorithms/326558

An Approach to Distinguish Between the Severity of Bullying in Messages in Social Media
Geetika Sarnaand M.P.S. Bhatia (2016). *International Journal of Rough Sets and Data Analysis (pp. 1-20).*
www.irma-international.org/article/an-approach-to-distinguish-between-the-severity-of-bullying-in-messages-in-social-media/163100

Organizational Transparency
Gustavo de Oliveira Almeida, Claudia Cappelliand Cristiano Maciel (2018). *Encyclopedia of Information Science and Technology, Fourth Edition (pp. 754-764).*
www.irma-international.org/chapter/organizational-transparency/183787

Effective Cultural Communication via Information and Communication Technologies and Social Media Use
Androniki Kavouraand Stella Sylaiou (2018). *Encyclopedia of Information Science and Technology, Fourth Edition (pp. 7002-7013).*
www.irma-international.org/chapter/effective-cultural-communication-via-information-and-communication-technologies-and-social-media-use/184397

Application of Deep Learning Algorithm in Visual Communication for Learning Behavior
Guoli Zhou (2026). *International Journal of Information Technologies and Systems Approach (pp. 1-26).*
www.irma-international.org/article/application-of-deep-learning-algorithm-in-visual-communication-for-learning-behavior/411703