World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:3, No:2, 2009

Impact of Faults in Different Software Systems: A Survey

Neeraj Mohan, Parvinder S. Sandhu, and Hardeep Singh

***Abstract*—**Software maintenance is extremely important activity in software development life cycle. It involves a lot of human efforts, cost and time. Software maintenance may be further subdivided into different activities such as fault prediction, fault detection, fault prevention, fault correction etc. This topic has gained substantial attention due to sophisticated and complex applications, commercial hardware, clustered architecture and artificial intelligence. In this paper we surveyed the work done in the field of software maintenance. Software fault prediction has been studied in context of fault prone modules, self healing systems, developer information, maintenance models etc. Still a lot of things like modeling and weightage of impact of different kind of faults in the various types of software systems need to be explored in the field of fault severity.

***Keywords*—**Fault prediction, Software Maintenance, Automated Fault Prediction, and Failure Mode Analysis.

I. Introduction

SOFTWARE maintenance has gained immense importance and attracted a number of research scholars because it involves a huge amount of cost and efforts for any software system. System testing and fault detection has become most important process in software life cycle. Various fault prediction models may be analyzed and proposed so that fault may be detected at an early stage and lot of testing efforts can be saved thereof. Early detection of faults may be implemented by various methods such as: Using Developers Information, Defect Tracking System, Self Healing System, Multivariate Analysis, Fault Injection etc.

II. Literature Survey

According to Lyu in [1], system architectures based on a cluster of computers have gained substantial attention in recent years. In such a system, complex software, hardware, operating systems, and application software need to be integrated for high system availability and data integrity. The performance and cost of the system can be greatly reduced by the use of separate error detection hardware and dedicated software fault tolerance methods. The application software may be used for the error detection, subsequent recovery actions and data backup. The application can be made as reliable as the user requires, being constrained only by the upper bounds on reliability imposed by the clustered architecture under various implementation schemes. Reliability modeling and analysis of the clustered system is presented by defining the hardware, operating system, and application software reliability techniques that need to be implemented to achieve different levels of reliability and comparable degrees of data consistency. According to [1], Markov reliability model may be used to capture these faults and subsequent recovery routines. It is also demonstrate how this cost-effective fault tolerant technique can provide quantitative reliability improvement within applications using clustered architectures.

Ostrand et al discusses in [2] that in a large software system some files that make the system particularly likely to contain faults are identified. Then a statistical model is developed that uses historical fault information and file characteristics to predict which files of a system are likely to contain the largest numbers of faults. In that method tester may easily detect the maximum number of faults without wasting much time. Automated tool may be produced that mines the projects defect tracking system. It can be used by testers without requiring any particular statistical expertise or subjective judgments. In that approach, system tester is major entity for fault detection [2].

Erdil describes in [3] that software development includes Requirements Engineering, Architecting, Design, Implementation, Testing, Software Deployment, and Maintenance phases. Maintenance is the last stage of the software life cycle. There are four major problems that can effect the maintenance process: unstructured code, maintenance programmers having insufficient knowledge of the system, improper documentation and software maintenance having a bad image. Maintenance process consists of four different parts:- Corrective maintenance that deals with fixing bugs in the code, Adaptive maintenance that deals with adapting the software to new environments, Perfective maintenance that deals with updating the software according to changes in user requirements and preventive maintenance that deals with updating documentation and making the software more maintainable. Maintenance process involves a lot of efforts in terms of developers time and development cost [3].

In [4] some improvements are proposed in the software

Neeraj Mohan is working as Assistant Professor in Deptt. Of Computer Science & Engg. Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 INDIA

Dr. Parvinder S. Sandhu is Professor with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 INDIA

Dr. Hardeep Singh is working as Professor with Computer Science & Engineering Department, Guru Nanak Dev University, Amritsar, India.

maintenance standards in paper. A new maintenance maturity model is proposed for software maintenance activities. This new model preserves a structure similar to that of the CMMI maturity model. It is based upon programmers experience, international standards and the literature available on software maintenance. Scope and architecture of the model is presented. The managerial problem of software maintenance is more dominant as compared to technical problems [4].

In a competitive world of fast paced software development, managers need to optimize the usage of their limited resources to deliver quality products in a strict time limit and allotted budget. In paper [5], an approach (The Top Ten List) which highlights to managers the ten most susceptible subsystems (directories) to have a fault is presented. To validate this work, this approach is applied to six large open source projects (three operating systems: NetBSD, FreeBSD, OpenBSD; a window manager: KDE; an office productivity suite: KOffice; and a database management system: Postgres).

According to Wilfredo Torres-Pomales [6], it is not possible to produce software without errors. The main cause of software design errors is the complexity of the systems. A brief overview of the software development processes, made by the paper, shows that it is hard-to-detect design faults. Faults are likely to be introduced during development phase. Component reliability is an important quality measure for system level analysis, software reliability is hard to characterize and the use of post-verification reliability estimates remains a controversial issue. Software safety and software reliability are two different issues, the comparative importance of these issues depends upon the nature of application. Multi-version techniques are based on the assumption that software built differently should fail differently. It indicates that if one of the redundant versions fails, at least one of the others should provide an acceptable output [6].

An attempt is made to make fully automated fault prediction systems that do not require any statistical expertise for user in fault prediction in [7]. Developer information is used for fault prediction. To make predictions a number of different characteristics and change history of project is considered. This information considerably increases the quality of next release of any software system [7].

According to Ohlsson in [8], software quality can be improved considerably if fault-prone modules can be detected at an early stage. The primary goal of the research by Ohlsson was to develop and refine techniques for early prediction of fault-prone modules. Before this work principal component analysis (PCA) and discriminant analysis (DA) approaches were used for building prediction models for faulty modules. Instead of dividing modules into fault-prone and non-fault-prone, the modules are categorized into several groups according to the ordered number of faults.

In [9] it is described that software fault detection and fault correction processes are two different processes. But these are related together. These are supposed to be studied together. A practical approach is to apply software reliability growth models to model fault detection, and fault correction process is assumed to be a delayed process. According to [9], in the artificial neural networks model, as a data-driven approach, tries to model these two processes together with no assumptions.

Houten explained in [10] that it is not feasible to build digital product models for maintenance purposes only. But if any digital product model is available, it may be used to support many maintenance-related activities which are very important in software life cycle. Some examples are: Product life cycle simulation, deterioration analysis, Failure Mode Effect Analysis (FMEA), failure diagnosis, maintenance ergonomic analysis etc. At the University of Tokyo, a Virtual Maintenance System has been developed to support some of the activities mentioned above. So, in future, CAD systems may be used to support product life cycle issues right from the start of the design process [10].

Catal elaborates in [11] that software testing is a time-consuming and expensive process. Quality expectations of software system are increasing in strict time limits. Software fault prediction models gives answer to such problems. These models can reduce the testing duration, project risks, resource and infrastructure costs. In this study, a novel fault prediction model is proposed to improve the testing process. This model provides a test strategy by focusing on fault-prone modules only. The goal is to predict the classes that will contain maximum faults in an Object-Oriented System. It will improve the next release of software system considerably [11].

There are several research papers that attempt to predict risk and fault prediction by analyzing the source code and applying a quantitative model. The accuracy of these models is significantly enhanced if process data is included in the development of the quantitative model for risk analysis.

Process Data may include the data that is gathered in and by the problem tracking system and the configuration management system. It is likely to include things like:

- Number of changes since last release
- Number of faults found since last release
- Number of different developers who turned over versions of this module since last release
- Number of features that were added that affected this module

Process data may also include things like experience levels of the developers, the amount of time that the module spent in review, the number of defects found in reviews, the number of test cases (and unique test cases) run that touched the module [12].

The problem of increasing software maintenance costs is addressed in [13]. A stochastic decision model is developed for the maintenance of information systems. Based on this modeling framework, optimal decision rule is derived for software systems maintenance. It also presents sensitivity analysis of the optimal policy. This model is applied to a large telecommunications switching software system. This modeling framework also allows for computing the expected time to perform major upgrade to software systems [13].

According to Li et al [14], the high cost of the federated structure, traditionally used in avionics systems, has led avionics systems suppliers to search for alternatives. The Integrated Modular Avionics (IMA) structure is an option because of its ability to share common hardware and software. However, sharing such resources results in new safety challenges—the complexity caused by sharing common hardware and software requires more complete and rigorous safety analysis. In this paper, a model-based fault-analysis technique for IMA systems is presented. The model accommodates multi-valued logic to handle the uncertainty of fault propagation inside an IMA system. An algorithm is also presented for fault propagation and root-cause analysis of IMA systems, based on this model. A visual-analysis tool is also presented that automatically performs fault-analysis tasks based on this model and algorithm [14].

The [15] presents a generic modeling framework to facilitate the development of self-healing software systems. A self-healing software system is capable of detecting the wrongs and it knows how to correct them. A model-based approach is used to categorize software failures into different categories. According to [15], self-healing is then achieved by transforming the model of the system into platform specific implementation instrumented with failure detection.

Hoffmann explains in [16] that the availability of software systems can be increased by preventive measures which are triggered by failure prediction mechanisms. In the paper two non-parametric techniques are employed which model and predict the occurrence of failures as a function of discrete and continuous measurements of system variables. Two modeling approaches: an extended Markov chain model and a function approximation technique utilizing universal basis functions (UBF) are employed. The presented modeling methods are data driven rather than analytical and can handle large amounts of variables and data. Both modeling techniques are applied to real data of a commercial telecommunication platform. The data includes event-based log files and time continuously measured system states. Results are presented in terms of precision, recall, F-Measure and cumulative cost. By using the presented modeling techniques the software availability may be improved by an order of magnitude [16].

Dependability modeling and evaluation (encompassing reliability and safety issues) of the two major fault tolerance software approaches-recovery blocks (RBs) and N-version programming (NVP)-are presented in [17]. The study is based on the detailed analysis of software fault-tolerance architectures able to tolerate a single fault. For each approach a detailed model based on the software production process is established and then simplified by assuming that only a single fault type may manifest during execution of the fault-tolerant software and that no error compensation may take place within the software. The analytical results obtained make it possible to identify the improvement, compared to a non-fault-tolerant software, that could result from the use of RB and NVP and to determine the most critical types of related faults [17].

In paper [18], the different predictor models are applied to NASA five public domain defect datasets coded in C, C++, Java and Perl programming languages. Twenty one software metrics of different datasets and Java Classes of thirty five algorithms belonging to the different learner categories of the WEKA project have been evaluated for the prediction of maintenance faults and severity. The results of validation are recorded in terms of Accuracy, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) for different project datasets [18].

In the survey performed in [19], the results about software faults encountered during the testing phase of a large real-time system are elaborated. The survey was conducted in two parts: the first part surveyed all the faults that were reported and characterized them in terms of general categories; the second part resurveyed in depth the faults found in the design and coding phases. These faults are analyzed and characterized so that these can be prevented in future [19].

According to Randell [20], any individuals and organizations as well have become dependent on computer based systems. So, there has been an ever-growing amount of research to improve the dependability of these computer based systems. In particular there has been much work on trying to gain maximum understanding of the various types of faults that need to be prevented, tolerated or corrected in order to reduce the probability and severity of system failures. In this regard various assumptions are to be considered such as faults prediction, fault tolerance and identification of faulty modules.

III. Conclusion

It is concluded from the above discussion that a lot work is going on in the field of software maintenance. Various fault prediction techniques and prediction models are explored to minimize the testing efforts. Enough research work has been done and going on in the field of software maintenance and early fault prediction. Still a lot of things like modeling and weightage of impact of different kind of faults in the various types of software systems need to be explored in the field of fault severity.

References

[1] Michael R. Lyu and Veena B. Mendiratta, "Software Fault Tolerance in a Clustered Architecture:Techniques and Reliability Modeling", Proceeding of IEEE Aerospace Conference, 1999. Volume 5, 1999, pp. 141 - 150.

[2] Thomas J. Ostrand and Elaine J. Weyuker, "A Tool for Mining Defect-Tracking Systems to Predict Fault-Prone Files", 1st international workshop on mining software repositories, 2005, pp. 85-89.

[3] Kagan Erdil, Emily Finn, Kevin Keating, Jay Meattle, Sunyoung Park and Deborah Yoon: "Software Maintenance As Part of the Software Life Cycle" Comp180: Software Engineering Project, December 16, 2003

[4] Alain April1, Jane Huffman Hayes and Reiner Dumke, "Software Maintenance Maturity Model (SMmm):The software maintenance process model", Journal of Software Maintenance 17(3), 2005, pp. 197-223.

[5] Ahmed E. Hassan and Richard C. Holt, The Top Ten List: Dynamic Fault Prediction, Proceedings of ICSM 2005: International Conference on Software Maintenance, Budapest, Hungary, Sept 25-30, 2005.

[6] Wilfredo Torres-Pomales, Software Fault Tolerance: A Tutorial , october-2000, URL: citeseer.ist.psu.edu/385206.html.

[7] Elaine J. Weyuker, Thomas J. Ostrand and Robert M. Bell, "Using Developer Information as a Factor for Fault Prediction", International Conference on Software Engineering, Proceedings of the Third International Workshop on Predictor Models in Software Engineering, 2007, pp. 8-18.
[8] Niclas Ohlsson, Ming Zhao and Mary Helander, "Application of multivariate analysis for software fault prediction", Journal of Software Quality Control, Volume 7 , Issue 1, 1998, pp. 51 - 66.
[9] Q.P. Hu, M. Xie and G. Levitin, "Robust recurrent neural network modeling for software fault detection and correction prediction", Reliability Engineering and System Safety, 92, no. 3, 2007, pp. 332-340.
[10] F.J.A.M. van Houten and F. Kimura, "The Virtual Maintenance System: A Computer-Based Support Tool for Robust Design, Product Monitoring, Fault Diagnosis and Maintenance, Annals of CIRP, vol. No.1, 2000, pp.91-94.
[11] Cagatay Catal and Banu Diri, "Software Fault Prediction with Object-Oriented Metrics Based Artificial Immune Recognition System", PROFES 2007, LNCS 4589,2007, pp. 300–314
[12] Greg Kaszycki, "Using Process Metrics to Enhance Software Fault Prediction Models", The 10th symposium on Software Reliability Engineering (ISSRE 1999), Boca Raton, Florida, Nov. 1-4, 1999.
[13] Krishnan, M.S., Mukhopadhyay, Tridas, Kriebel and Charles H., "A decision model for software maintenance" Information Systems Research, Vol. 15, No. 4, December 2004, pp. 396-412.
[14] Wanchun Li, Heena Macwan and Mary Jean Harrold, "Model-based Fault Analysis for Avionics Systems", 1st International Workshop on Aerospace Software Engineering (AeroSE 07),May 21-22, 2007, Minneapolis, USA.
[15] Michael Jiang, Jing Zhang, David Raymer and John Strassner, " A Case Study: A Model-Based Approach to Retrofit a Network Fault Management System with Self-Healing Functionality, ECBS 2008, pp. 9-18.
[16] Günther A. Hoffmann, Felix Salfner, Miroslaw Malek, "Advanced Failure Prediction in Complex Software Systems", Advanced Failure Prediction in Complex Software Systems, April 2004
[17] Arlat, J. Kanoun, K. Laprie, J.-C, "Dependability modeling and evaluation of software fault-tolerant systems", IEEE Transactions on Computers,Volume 39 , Issue 4,1990, pp. 504 - 513.
[18] Parvinder Singh Sandhu, Sunil Kumar and Hardeep Singh, "Intelligence System for Software Maintenance Severity Prediction", Journal of Computer Science, 3 (5), 2007, pp. 281-288.
[19] Dewayne E. Perry and Carol S. Stieg, "Software Faults in Evolving a Large, Real-Time System: a Case Study", In 4th European Software Engineering Conference ESEC93,1993, pp. 48-67.
[20] Brian Randell, "Facing Up to Faults", The Computer Journal, Vol. 43, January 31, 2000.