Abstract— Reliability is often measured as the probability of failure occurrence, frequency of failures, or in terms of system availability. All the real time systems tend to have faults though not all of them will result in failure. In order to have highly reliable systems it is imperative to have failures monitored, tracked and kept under the tolerance limit. The goal of the fault life cycle activities are to remove errors and their effects from the computational state before a failure occurs. In certain cases, where the failure has already occurred, quick recovery is the next immediate target. This review intends to give overview about the techniques practiced and studied in the context of fault detection and diagnostics area and the current trends.

Index Terms— Fault detection; fault diagnosis; log; reliability; software fault; detection trends.

I. INTRODUCTION

Software reliability engineering has become very crucial in the recent days as complexity of the systems has grown exponentially with technological advancements. It is not only the application faults that results in failure but also many environmental factors due to the fact that every system today is dependent on many other systems and interfaces. Though the impact of such failures vary with the nature of systems, many of them have resulted in customer dissatisfaction and loss of business [1]. Hence reliability of a system is to be conceptualized from the scratch, required measures are to be embedded in all the development lifecycle phases and controlled throughout its existence.

Software reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time. [2]. In order to achieve highly reliable systems, the fault life cycle areas to be considered are classified into four areas namely fault prevention, fault removal, fault tolerance and fault forecasting[3]. Fault prevention and removal are primarily handled in the pre-release phases of an application mostly by review and testing processes. Fault tolerance is about providing the ability to the system to withstand failures. Fault forecasting helps us understand the remaining faults in the system and also the likelihood of future occurrence. We focus our discussion to fault detection using various techniques and diagnosis of faults before or immediately after their occurrence and look at the recent trends of fault detection in latest technologies. Fault detection and diagnosis in this paper focuses on dealing with post release context though many of them may be relevant in construction phases as well. Section II provides an abstraction of techniques used widely in various research activities listed in Section III. Section III gives a brief of the study and the result. Section IV gives a brief on recent technologies and how fault detection techniques have evolved for them.

II. FAULT DETECTION TECHNIQUES

The two types of detection strategies applied are static and dynamic. Static detection involves automated tools running through the code and data to identify defects and abnormalities. They are usually based on configurable rules and predefined specification on expected values. Dynamic detection involves techniques such as pattern matching, studying of traces and log files at run time. While static analysis does not require the programs under analysis to be executed, dynamic techniques need them mandatorily.
The detection activities are carried out at various life cycle phases of the project. Fig.1 gives a schematic representation of how they fit into project activities. The spectrum of solutions uses either one of these or a combination of some based on their need and criticality.

A. Automated Static Analysis (ASA)

Manual Code walkthrough is one of the oldest practices still being practiced but automated tools are increasingly being used especially for standard related problems. They detect defects related to non-compliance to standards, possible memory leaks, variable usage etc. They have a critical place in the development phase as they save significant rework effort and defect leakage in testing cycles. PMD, CheckStyle and Findbugs are some of the commonly used tools in Java technology and many such tools exist in all the technologies. Though this plays a major role in the development cycle it is not practiced so widely in maintenance. However, for systems having ASA tool compliant source, this can be used as a hygiene factor and good detection mechanism as any bug introduced at the field is highly expensive. At maintenance cycle ASA tools may not find many faults that can result in failures. A study on the effectiveness of ASA Tools on open source code reveals that it finds less than 3% of the faults[4]. Even meager number in such case is highly valuable considering the negligible effort spent on this.

B. Graph mining

This is a dynamic control flow based approach that helps identifying faults that may be non-crashing in nature. It uses call graphs which are reduced for simplicity in processing. The graph node represents functions and calls to other function is represented by edges. Edge weights are introduced based on call frequencies. Variation in call frequency and variation in call structure are potential faults. If there are any issues in the data being passed between methods that could also affect the call graph due to its implications.

C. Classifiers

A decision tree, clustering algorithm or a neural network may be used to identify abnormal events from normal events. There are also classifiers which are trained by labeling the faulty runs when a defect is observed. Some of the commonly used classifiers are NaiveBayes and Bagging. The Bayesian Classification represents a supervised learning method as well as a statistical method for classification. It assumes an underlying probabilistic model and it allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes.

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.

A recent research work [5] done on this area proposes a secondary unsupervised model that captures the probability distribution of each code region’s normal behavior, making it possible to identify the events when it behaves abnormally. This abnormality information is used to filter the labeling presented to the classification algorithm to focus it on the abnormal observations.

D. Pattern Mining

This is also classifier based but utilizes unique iterative patterns for classifying sequential data using the software trace analysis for failure detection. A set of discriminative features capturing repetitive series of events from program execution traces are first executed. Subsequently feature selection is done to select the best features for classification. The classifier model is trained with these set of features which will be used in identifying the failures.

E. Rule Based Logging

The approach suggests automated log insertion in a structured way driven by certain rules. The important locations for logging are function start and end, start and end of an entity life cycle, interaction start and end. Any abnormality on this flow to be automatically detected and alerted.

III. FAULT DIAGNOSIS

The primary input methods used for fault diagnosis are core dumps and log files. Core dumps are generated only when there is a crash and contains the memory image during crash and often difficult to analyze. Significant research projects have been done to improve the log files to present more
relevant information and also read automatically and alert users for any abnormalities.

A. Log Causally Related Information

Each point of logging is identified and the control flow and data flow is tracked till that point within the function. The variables involved in the condition and each memory location involved are the information of interest to the support engineer who is involved in analyzing the fault. All such information is suggested to be added to the same log point.

B. Visual representation of logs

Log file information is usually disconnected and it takes huge time for developers to understand correlating the related. Simple technique is to relate messages based on the message type. There is a fixed relation between message type and the set of identifier classes referenced by messages of that type. The graph of this relation is the identifier graph. Every identifier class and message type corresponds to a graph node. There is an edge between the nodes corresponding to an identifier class and a message type if the messages of that type include identifiers of that class. Edges used here are undirected. These graphs help developers spot several related classes of deficiencies in logs as well as identifiers that are not present or ambiguous.

C. Random Indexing and Support Vector Machine

SVM is a commonly used mining technique which can be used to predict faults from log files. Random Indexing is used to represent sequences of operations extracted from log files. Support vector machine is used to classify them as either failure or success. This approach holds good due to the sequential nature of system messages. To improve the accuracy of non failure messages weighted SVMs are used.

D. Spectrum Based Fault Localization

Spectrum based fault localization techniques can be used to locate a fault in an erroneous piece of code when it is known that some specific program is failing. This is an automated technique which can be applied on a program without much knowledge about the program hence suits best for the testing schemes but can be used in other phases as well. This is a dynamic technique that involves running a program multiple times with various inputs. A matrix is created with the results indicating either success or failure. This is compared with a block hit matrix and similarity coefficients arrived at for each block. Blocks represent blocks of code within a program some of them subset of one or more other blocks. Similarity coefficients are commonly used in clustering to identity degrees of resemblance. This highest similarity coefficient helps locating the smallest block of code that contains the fault.

IV. WORK DONE

Wedyan et al. have carried out an empirical evaluation[4] on three open source software with three ASA Tools. The study showed that ASA tools would minimally help to detect faults that could be reported in future. They help lot in writing well documented code and identifying refactoring modifications.

Frank Eichinger et al. proposed a novel reduction technique [6] for call graphs which introduces edge weights. Edge weights represent call frequencies. An entropy based algorithm has been introduced to do scoring of numerical edge weights and frequent sub graph mining. The traditional and entropy based approaches are used in hybrid way which found high number of faults.

Bronevetsky, G. et al. has demonstrated the ability [5] to detect and characterize faults with 85% accuracy with a statistical classifying models on which improvement done by filtering the inputs with the knowledge on system abnormality learnt with unsupervised model. The naive approach provides a classification algorithm with execution data that labels each event with the type and location of any fault occurring at the time, if any. The approach then applies the classifier to events from a new execution to label events. Labels that indicate faults, including the fault type and location, are presented to the administrator for further review. This approach has low accuracy and frequently reports many events that correspond to the same fault, which can overwhelm system administrators. In contrast, the suggested approach first applies an unsupervised model to identify the events that are abnormal and removes the fault labels from all other events. It then presents this filtered training set to the classification algorithm, which results in improved accuracy. Finally all the fault detections are aggregated to provide just one notification for each system fault, thus reducing the reporting to system administrators. The classifier’s attention is kept on the low probability events and improves the fault detection and classification accuracy from 12% to 85% on faulty runs. The false positive rate is maintained as low as 5%.

A.A. Shahrjoo Haghighi et al. have studied 37 different classifiers in fault detection systems and reported bagging classifier to provide the best performance among them[7]. Leo Breiman defines bagging predictors [23] as a method of generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages the multiple the versions when predicting a
class. For performance evaluation two indicators have been used namely accuracy and AUC (Area under curve).

David Lo et al. mines a set of discriminative features capturing repetitive sequences of events from program execution traces [8]. The technique initially mines a set of discriminative execution trace. Subsequently feature selection is done to select the best features for classification. The classifier model is trained with these set of features which will be used in identifying the failures. Experiments and case studies on traces of several benchmark software systems and a real-life concurrency bug from MySQL server show the utility of the technique in capturing failures and anomalies. This pattern based classification technique is found to outperform the baseline approach by 24.68% in accuracy.

M. Cinque, Pecchia. A et al. as part of their study [9][10] analyzed the logs to find their effectiveness in identifying faults. The study was carried out on three widely used open source software. Fault operators were defined and used by a fault injector framework. Around 60% of the faults did not leave any errors in the log file. It is also an interesting finding that software systems are prone to log errors that happens with OS and interfaces that with erroneous algorithm. Later as a continuation [11], a rule based approach has been defined. Rules were defined on where to insert logging code, certain errors based on time-out mechanism are automatically detected and logged by a logger framework[16]. The logging framework is completely decoupled from the log invocation code hence performance is safeguarded.

Ding Yuan et al. have presented a tool called Log Enhancer [12] which enhances every log message in the target application to collect causally related information. Log Enhancer can collect historic, intermediate information prior to failures and also provide diagnostic information. Log Enhancer’s source code analysis is implemented using the Saturn static analysis framework. Saturn models C programs precisely and allows user to express the analysis algorithm in a logic programming language. It is summary-based, meaning it conducts its analysis for each function separately and then generates a summary for each function. At the calling sites of a function, the summary is used instead of going deep into the function. As part of the analysis, it was found that 95% of the variables included in the log messages by developers overtime can be automatically identified by Log Enhancer.

Ariel Rabki[13] et al. have used identifier graph developed from the graph which helps understanding how entities are related. The approach is to construct an abstract graphical representation of application logs and to assess the logs by analyzing and reasoning about the model. It helps developers spot several related classes of deficiencies in logs: identifiers that are absent, inconsistent, or ambiguous. Each of these defects imply a relationship between system entities that was determinable at runtime, but which was not recorded in the logs and which cannot be easily reconstructed afterwards.

Ilenia Fronza et al. [14] applied RI to represent sequences: each operation is characterized in terms of its context. SVMs associate sequences to a class of failures or non-failures. Weighted SVMs are applied to deal with imbalanced datasets and to improve the true positive rate. The approach was to log files collected during approximately three months of work in a large European manufacturing company. The result showed that weighted SVMs sacrifice some specificity to improve sensitivity. Specificity remains higher than 0.80 in four out of six analyzed applications.

Peter Zoeteweij et al. [15] have used Spectrum based fault localization technique to locate the erroneous code in a program. Experiment has been carried out in a consumer electronics product and found to be suitable.

V. CURRENT TRENDS

The research on emerging technology[17][18] and analysis on them indicates that Business Intelligence, Infrastructure as a service and mobile applications and platforms are bringing in more evolutionary changes and the business investments will be on the direction of the big four namely cloud, mobile, social and data.

Cloud computing has attracted more interest recently. More and more internet applications are moving towards cloud. The fault detection is more complex and difficult in cloud computing system due to its dynamic nature. Ying Jiang et al, as part of [19] suggests classification models for fault detection. In order to deal with the faults in cloud computing systems, discusses four important aspects and proposes an improved C4.5 algorithm to detect the fault. This algorithm is based on decision trees.

Guisheng Fan et al. [20] proposed a model based Byzantine fault detection technique for cloud computing. A cloud computing fault net (CFN) is used to precisely model the different components of cloud computing. Service resources, cloud module, the detection and failure process are various components involved. Once the model is constructed using the components it analyzed. The fault detection strategy suggested based on this, which can dynamically detect the fault of cloud application in the execution process.

Inability of wireless protocols to detect and respond to faults in real-time impacts is a current area of concern. Real-time fault detection in mobile computing enables the system to respond to fault indicating situations within certain time constraints. [21] surveys the literature on fault detection
and tolerance schemes used in mobile networks. The survey reveals that most of the currently existing algorithms do not scale well for the needed real-time constraints and is an evolving area of research. Recently Yin Zhang et al, proposed a detection model [22] based on a linear prediction algorithm and the normalization process of the prediction deviation makes the model more simple and flexible to use. It has been tested in a simulated LTE environment, and the results indicate that the model can indeed detect real system faults while tracking the normal variations of the Key Performance Indicators(KPIs) of the network.

VI. CONCLUSION

Software reliability started getting increased attention nowadays and having highly fault tolerant system is an implicit need today. In this paper, research work on fault detection techniques, diagnostics as well as the recent trend on latest technologies with respect to fault detection has been discussed. There are vast number of methods and techniques used to detect faults and diagnose them in software systems but not every technique suits every system. Selection of such technique is driven by factors such as the reliability goal, technology platform, criticality of the system, adaptability and size and complexity of the system. At the high level the trend is leaning more towards statistical models and mining techniques in hybrid way for automated detection and trace oriented solutions for diagnosis for traditional systems. Modern day applications fault handling are in the early stages of the research and the solutions try to build tolerance at the architecture level as much as possible.

REFERENCES


[8] David Lo, Hong Cheng, Jiawei Han, SiyuCheng Khoo and Chengnian Sun, "Classification of Software Behaviors for Detection: A Discriminative Pattern Mining Approach," KDD '09 Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, Pages 557-566 ACM New York, NY, USA ©2009.


K. Umadevi received B.E degree from Govt.College of Technology, Bharathiar University and pursuing M.Tech in Computer Science at Bharath University. She has more than 15 years of experience in the field of Information Technology with exposure to service delivery, product development and release. She was a former member of Project Management Institute and has managed and delivered successful software projects for more than a decade as part of leading IT organizations including Infosys and Cognizant. She has vast experience in all the software development life cycle phases spanning Requirements analysis, Architecture definition, Design, Development, Testing and Implementation with financial services and project management domain. Her expertise includes various technologies with special emphasis on Object oriented analysis, design and testing.

S. Brintha Rajakumari received MCA in 2001 from the Madurai Kamaraj university and ME from Sathyabama University in the year 2011. She is working as an Assistant professor at Bharath University for last seven years. She has more than 12 years of experience in academic and research in areas of interest being Database, Data Warehousing and Mining and Cloud Computing. She has published over 10 research papers in international and national journals of repute. She is pursuing her Ph.D in the field of Database Management System.