ISSN:0975-9646

Improving Software Quality Using FMEA and FTA Defect Prevention Techniques in Design Phase

Shahin Fatima, Dr.Mohd. Rizwan Beg, Shadab Siddiqui

Department of Computer Science and Engineering,
Integral University, Lucknow (India)

***Abstract*— The cost of finding and correcting defects represents one of the most expensive software development activities. And that too, if the errors get carried away till the final acceptance testing stage of the project life cycle, then the project is at a greater risk in terms of its Time and Cost factors. A small amount of effort spent on quality assurance will see good amount of cost savings in terms of detecting and eliminating the defects. The purpose of defect prevention is to identify those defects in the beginning of the life cycle and prevent them from recurring so that the defect may not surface again. Software for safety-critical systems must deal with the hazards identified by safety analysis in order to make the system safe, risk-free and fail-safe. Certain faults in critical systems can result in catastrophic consequences such as death, injury or environmental harm. The focus of this paper is an approach to software safety analysis based on a combination of two existing fault removal techniques. A comprehensive software safety analysis involving a combination of Design Failure Modes and Effects Analysis (DFMEA) and Design Fault Tree Analysis (DFTA) is conducted on the functions of the critical system during design phase to identify potentially hazardous design faults. A prototype safety-critical system - Elevator Door Control System (EDCS), is described here and DFMEA and DFTA technique is applied on a component of EDCS.**

Keywords*—Defect, Defect Analysis, *Defect Prevention*, Root Cause Analysis software safety, safety-critical systems, *DFMEA, DFTA

I. INTRODUCTION

Software Defect can be defined as “A *software defect* is a deficiency in a software product that causes it to perform unexpectedly”. From a software user’s perspective, a defect is anything that causes the software not to meet their expectations. In this context, a *software user* can be either a person or piece software. Defect Prevention (DP) is a process of improving quality whose purpose is to identify the common causes of defects, and change the relevant process (es) to prevent that type of defect from recurring [5]. DP also increases the quality of software product. Defect prevention firstly involves identification of defect, and then modification and changing the relevant processes, preventing the re-occurring of the defects in the development process. As early as defects are identified in the development process, the more smoothly the development process progresses. In this paper we have discussed two defect prevention techniques viz. DFMEA and DFTA. The first defect prevention technique is Design failure modes and effects analysis (DFMEA). This technique helps product teams anticipate product failure modes and assess their associated risks in design phase of software. Prioritized by potential risk, the riskiest failure modes can then be targeted to design them out of the software or at least mitigate their effects. The second defect prevention technique described is fault tree analysis (FTA). Unlike failure modes and effects analysis, which focuses on potential failure modes and does not drill deeply into the potential causes, fault tree analysis starts with an “unintended event” (for example, a defect or failure mode) and then drills into all the potential causal events. This makes it a natural complement to DFMEA.

II. WORK FLOW STAGES DEFECT HANDLING

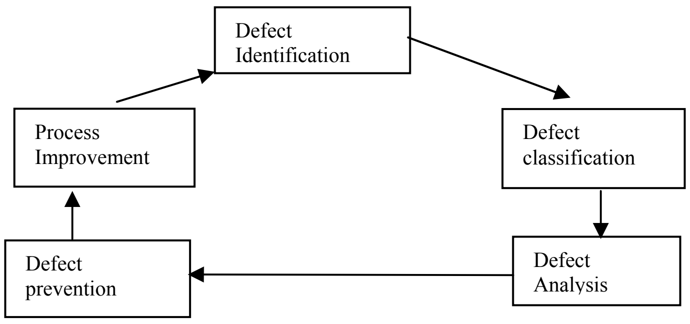


Figure 1: Process Improvement Workflow

A Defect Identification OR Defect Detection in Software Process

Defect Identification is the first activity involved for improving the quality of the Software Process. It is widely used in many of the Software projects, for discovery of the Software Defects, then documenting them for improving the quality of the Software product [5].

B Defect Classification

ODC classifies defect at two different points in time: One is Opener Section, where the defect were firstly investigated and second one is Closer Section, where the defects are fixed. For Small Sized and Medium Sized Projects defects are classified to first level of ODC to save efforts and time. For larger projects defects are deeply understood and analysed [5].

C Defect Analysis

By the term analysis we meant the identification of the root cause of the defect and then further devising the solution to overcome the defect in further development process which will be further useful in improving the software quality and productivity of the software project. Some of the defect analysis techniques such as Fish Bone Analysis, Defect Classification and using defect taxonomies and the Root Cause Analysis (RCA). RCA goal is to first identify the root cause of the defects and then initiating actions for the defect elimination [5].

D Defect Prevention

The primary goal of defect prevention is to anticipate and prevent defects proactively before they can occur and cause failures or confuse users. This is the best approach because a defect that does not occur is also a defect that need not be caught, fixed, and supported. The savings resulting from the successful application of defect prevention techniques can be reapplied elsewhere in the product cycle. Examples of defect prevention techniques used in this paper are Design failure modes and effects analysis (DFMEA), Design fault tree analysis (DFTA) [5].

E Process Improvement

By the term Process improvement we mean the continuously working for improvement for the quality of the software process. Process Improvement meant that following preventive actions for software improvement and then further taking actions for further improvement of quality. By continuous process improvement we identify the errors continuously, correct it and hence the quality of software is also improved [5].

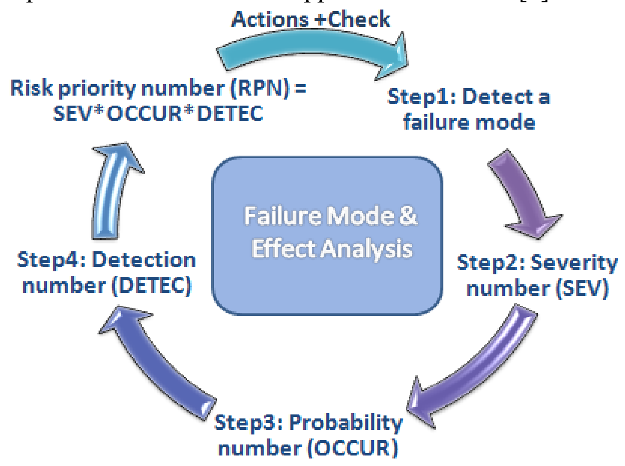
III. Implementation of Defect Preventive (DP) Techniques

The first defect prevention technique is failure modes and effects analysis (DFMEA).

A. Design Failure Modes and Effects Analysis

Design Failure modes and effects analysis (DFMEA) is a defect prevention technique used to identify potential failure modes in a product design phase, assess the risk of each potential failure, and then implement appropriate actions to eliminate or mitigate those failure modes. Once identified, this information can be persisted and used in future projects to help avoid defects.

The purpose of DFMEA is to identify possible failure modes of the system components during design phase, evaluate their influences on system behaviour and propose proper countermeasures to suppress these effects [4].


Actions +Check
Risk priority number (RPN) = SEV*OCCUR*DETEC
Step1: Detect a failure mode
Failure Mode & Effect Analysis
Step4: Detection number (DETEC)
Step2: Severity number (SEV)
Step3: Probability number (OCCUR)

1) Procedure

Step 1: Identify and Describe the Target Product Focus Area

Step 2: Create a DFMEA Worksheet and Enter Initial Data

Step 3: Determine Failure Modes and Add to DFMEA Worksheet

A *failure mode* is a type of failure that could occur. In software systems, this is evidenced by symptoms such as a blue screen, system hang, incorrect output, and data corruption. Identifying Failure Modes Potential failure modes can be identified from many different sources:

- ■ Brainstorming
- ■ Root cause analysis
- ■ Defect taxonomy

Step 4: Rate Failure Mode Impact, Likelihood, and Detectability

Step 5: Calculate the Risk Priority Number for Each Failure Mode.

The *risk priority number (RPN)* is a very straightforward calculation. It is simply the product of the impact rank, likelihood rank, and delectability rank:

*RPN = Impact Rank * Likelihood Rank * Delectability Rank.*

Step 6: Identify the Failure Modes with the Highest Potential Risk

Step 7: Define an Action Plan to Eliminate or Mitigate the Causes

Step 8: Reassess the Risk Priority After the Actions Are Implemented

2) DFMEA BENEFITS:

Design Failure modes and effects analysis is a procedure for proactively identifying potential failures and assessing their risks. In software development, this provides benefits such as the following:

- ■ Improved software quality and reliability result in an improved customer experience and greater customer satisfaction.
- ■ Focus on defect prevention by identifying and eliminating defects in the software design stage helps to drive quality upstream.
- ■ Proactive identification and elimination of software defects saves time and money.
- ■ Prioritization of potential failures based on risk helps support the most effective allocation of people and resources to prevent them.

B. Design Fault Tree Analysis

Design Fault tree analysis (DFTA) is a technique that uses Boolean logic to describe the combinations of intermediate causal events that can initiate a failure (“unintended event”). Where Design failure modes and effects analysis (DFMEA) strives to enumerate all failure modes for a product and then estimate their risk, fault tree analysis starts with a specific failure and strives to enumerate all the causes of that event and their relationships. The overall goal is to identify specific opportunities to eliminate or mitigate the causes that can ultimately result in product failure. A fully constructed fault tree represents a failure and all its potential causes. From a qualitative perspective, the tree represents a logic diagram that depicts a set of causal event sequences. Ultimately, this diagram can be used to identify *cut sets* that are unique combinations of *basic* causal events for which, if each event occurs, the failure will occur. A cut set can potentially be reduced by removing basic events and still cause the failure. Ultimately, this reduction results in a minimal cut set of basic events that cannot be reduced further. These minimal cut sets can help software development teams identify the combinations of basic causal events that will result in product failure. Targeting and eliminating these basic events can prevent one or more failure opportunities and improve the overall reliability of the product. From a quantitative perspective, if the probability of occurrence for each causal event can be estimated, this information can be used to calculate the overall probability that the failure will occur. This is useful for software reliability analysis and can provide valuable input into failure modes and effects analysis, which depends on accurate estimates of cause likelihoods [4].

1) *Procedure:-*

Design Fault tree analysis is a deductive analysis technique that starts with a failure and focuses on deducing all the potential causes and their relationships. Therefore, DFTA starts with choosing a target failure, possibly identified in DFMEA, and then using the standard DFTA event and Boolean gate symbols to create the logic diagram of possible causal event sequences. After it is constructed, the fault tree can be analysed manually to identify the key causal events that can lead to the failure. Alternatively, specialized DFTA software can be used to quickly perform an_automated analysis of the fault tree. The following procedure describes the steps for completing a DFTA in more detail:-

Step 1: Select and Define the Failure to Analyse

Step 2: Create the Fault Tree

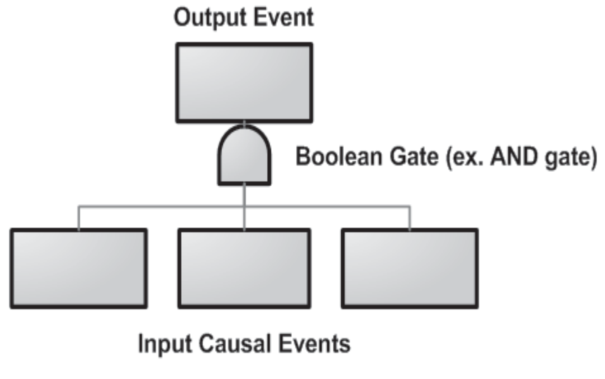


Fig: 2 - Basic fault tree construct

Step 3: Analyse the Fault Tree

The mathematical concept of cut sets originated in graph theory and has been used in the context of fault trees to mean the unique combinations of basic events that, should they all occur, will cause the failure or undesired event

Step 4: Review the Matrix Rows to Identify Minimal Cut Sets As a reminder, a minimal cut set is a cut set where no events can be removed and still cause the failure *if they all occur at the same time*.

Step 5: Interpret the Result

2) *DESIGN FAULT TREE ANALYSIS BENEFITS*

Fault tree analysis is a deductive analysis technique that starts with a failure and focuses on deducing all the potential causes and their relationships. In software development, this provides benefits such as the following:

- Improved software quality and reliability result in an improved customer experience and greater customer satisfaction.
- DFTA includes the capability of diagramming any pertinent causal events that can lead to failure, including software and hardware errors, human errors, and operational or environmental events.
- DFTA can be used proactively to understand and identify the causes that can lead to failure. This information can be used to prevent these causes.
- DFTA can be used reactively to diagnose and learn from a failure that has occurred, whether in testing or as part of final product usage.

IV. Elevator Door Control System (EDCS)

Elevator doors protect riders from falling into the shaft. The most common configuration is to have two panels that meet in the middle, and slide open laterally. In a cascading telescopic configuration (potentially allowing wider entryways within limited space), the doors run on independent tracks so that while open, they are tucked behind one another, and while closed, they form cascading layers on one side. This can be configured so that two sets of such cascading doors operate like the centre opening doors described above, allowing for a very wide elevator cab. Some buildings have elevators with the single door on the shaft way, and double cascading doors on the cab. During a failure of an ingress-egress control system, *e.g.*, a user propping a door open somewhere in a building, a fail-secure lock will close, lock, and remain locked even when a user attempts to unlock it with the key that the user usually employs. In such a case, an independent release, such as a reboot or disarming of the securing mechanism, is required. In contrast, a component may be considered fail-safe even if its failure does not secure the system. For example, if a door locked from the inside is left unlocked or is unlocked at the wrong time, it has failed (in some cases, along with the entire system), the door may be (but is not necessarily) fail-safe if its being unlocked does not open it or attract additional attention to its unlocked state[8].

V. Safety Analysis of EDCS

The safety analysis of ECS software functions takes place in three sequential steps [4].

A. *Design Failure Mode and Effects Analysis (DFMEA)*

A Design potential FMEA is an analytical technique utilized primarily by a design responsible engineer/team as a means to assure that, to the extent possible, potential Failure Modes and their associated Causes/Mechanisms have been considered and addressed. End items, along with every related system, subassembly and component, should be evaluated. In its most rigorous form, an FMEA is a summary of the team's thoughts (including an analysis of items that could go wrong based on experience) as a component, subsystem, or system is designed. This analysis is performed in order to determine the top events for lower level analysis. DFMEA analysis will be performed following the list of failure types encountered during design phase. DFMEA will be used to identify critical functions based on the applicable software specification. The severity consequences of a failure, as well as the observability requirements and the effects of the failure will be used to define the criticality level of the function and thus whether this function will be considered in further deeper criticality analysis. The formulation of recommendations of fault related techniques that may help reduce failure criticality is included as part of this analysis step [4].

B. *Design Fault Tree Analysis (DFTA)*

After determining the top-level failure events, a complete Design Fault Tree Analysis shall be performed to analyse the faults that can cause those failures in design phase. This is a top down technique that determines the origin of the critical failure. The top-down technique is applied following the information provided at the design level, descending to the code modules. DFTA will be used to confirm the criticality of the functions (as output from DFMEA) when analysing the design (from the software requirements phase, through the design) and to help:

- Reduce the criticality level of the functions due to software design fault-related techniques used (or recommended to be used)
- Detail the test-case definition for the set of validation test cases to be executed [4].

C. *Evaluation of Result*

The evaluation of the results will be performed after the above two steps in order to highlight the potential discrepancies and prepare the recommended corrective measures.

1) DFMEA Analysis of EDCS

The DFMEA, a sample of which is shown in the Table 1 below presents some software failure modes defined for EDCS. The origin and effects of each failure mode are analysed identifying the top level events for further refinement, when the consequence of this failure could be catastrophic for this system. The top events that were singled out for further analysis of failure mode are Improper Functionality, Inadequate timing of elevator door, incorrect selection of data structure and Resource management. Focus on defect prevention by identifying and eliminating defects in the software design stage helps to drive quality upstream

2) DFTA Analysis of EDCS

Fault tree analysis (FTA) is a technique that uses Boolean logic to describe the combinations of intermediate causal events that can initiate a failure (“unintended event”). fault tree analysis starts with a specific failure and strives to enumerate all the causes of that event and their relationships. The fault tree is a graphical representation of the conditions or other factors causing or contributing to the occurrence of the so-called top event, which normally is identified as an undesirable event. A systematic construction of the fault tree consists in defining the immediate cause of the top event. These immediate cause events are the immediate cause or immediate mechanism for the top event to occur. From here, the immediate events should be considered as sub-top events and the same process should be applied to them. All applicable fault types should be considered for applicability as the cause of a higher level fault. This process proceeds down until the limit of resolution of the tree is reached, thereby reaching the basic events, which are the terminal nodes of the tree.

Table 1 . Example of DFMEA table for software in design phase of EDCS

Failure Mode	**Causes of Failure**	**Consequences**	**Predicted Severity**	**Recommended Solution**
Improper functionality	1) incomplete requirement 2) Lack of info. with wrongly estimated objectives of project 3) Incorrect selected alternative for final solution 4) wrong technology used	Improper working of software which can lead to catastrophic failures	Critical	Software should be designed to work in proper functional mode
Inadequate Timing	1) Physical obstructions 2) delays 3) Bad decision making	Unpredictable sequence of operations leading to hazards	Critical	Designing should be such that it runs in proper order
Incorrect selection of Data Structure	1) Incomplete logic 2) Excessive type conversion 3) Incorrect Algorithm	Out of memory errors	High	Algorithm logic is Verified for accuracy. Data Structures and Memory overflow is checked.
Resource management	1) Low availability of funds 2) Inadequate time management 3)Improper Team harmony	Project can be delayed or it can lead to failure of project	High	Proper planning and execution of available resources

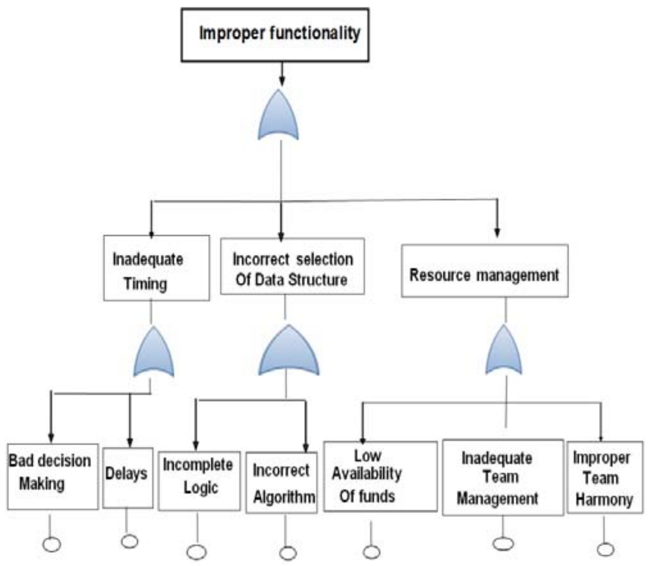


Figure 3. Design Fault Tree sample for top event

VI. Result & Analysis

In view of the comprehensive safety analysis, and specification and implementation the safety properties during EDCS design and development, the expected result was that safety-specific EDCS development would produce a software system with fewer latent safety-critical faults than traditional non safety specific techniques alone. This is due to the belief that the safety-specific techniques will prevent safety critical faults in the specifications and designs that the traditional techniques have a tendency to miss. During the operation of EDCS, the safety specific development version of EDCS clearly demonstrated the fulfilment of the safety properties. For example, if the functionality of door of the elevator is not proper then it can lead to various catastrophic hazards, so control program should be designed to work in proper functional mode. If the timing of door open/close is not adequate then it can lead to some unpredictable operations so designing should be such that it runs in proper order. Likewise, in the safety-version of EDCS, if the selection of data structure in design mode is improper then whole software program can go wrong so algorithmic logic must be verified for accuracy before implementing it.

VII. Conclusion

Implementation of defect preventive action not only helps to give a quality project, but it is also a valuable investment. Defect prevention practices enhance the ability of software developers to learn from those errors and, more importantly, learn from the mistakes of others. The benefits of adopting defect prevention strategy would be enormous and to list a few, Defect prevention reduces development time and cost, increases customer satisfaction, reduces rework effort, thereby decreases cost and improves product quality. This paper discussed a FMEA and FTA defect prevention techniques in design phase of software and applied this approach to software safety analysis for critical systems. A comprehensive software safety analysis involving a combination of DFMEA and DFTA techniques was conducted on a component of the critical system to identify potentially hazardous design faults. The safety properties of the prototype elevator door control system were identified as part of the safety critical requirements. We also briefly compared safety-specific and non-safety specific techniques at developing EDCS. The non-safety version of EDCS broadly focused on achieving the functional behaviour of the system. The safety-specific version clearly demonstrated that the software safety properties identified in EDCS specification were fully met in the working system.

Acknowledgement

I would like to thank Dr. Mohd Rizwan Beg HOD, Computer Science Dept, Integral University, Lucknow, India, for his valuable support, guidance and encouragement.

References

1. Failure knowledge diagnosis model based on the integration of Fmea and Fta.(IEEE)Print:-ISBN: 978-1-4673-1689 DOI: 10.1109/ICCIAutom.2011.6183941Date of Current Version: 16 April 2012 Issue Date: 27-29 Dec. 2011
2. Contemporary Trends in Defect Prevention: A Survey Report I.J.Modern Education and Computer Science, 2012,3, 14-20 Published Online April 2012 in MECS DOI: 10.5815/ijmecs.2012.03.02
3. The application of FMEA in the oil industry in Iran: The case of four litre oil canning process of Sepahan Oil Company (African Journal of Business Management Vol. 5(8), pp. 3019-3027, 18 April, 2011 DOI: 10.5897/AJBM10.1248 ISSN 1993-8233 ©2011 Academic Journals
4. FMEA and Fault Tree based Software Safety Analysis of a Railroad Crossing Critical System (Global Journal of Computer Science and Technology Volume 11 Issue 8 Version 1.0 May 2011 ISSN: 0975-4172 & Print ISSN: 0975-4350
5. Defect Analysis and Prevention for Software Process Quality Improvement International Journal of Computer Applications (0975 – 8887)Volume 8– No.7, October 2010
6. Fmea and Fta analysis for application of the reliability centered maintenance methodology: Case study on Hydraulic turbines ABCM Symposium Series in Mechatronics - Vol. 3 - pp.803-812
7. Improvising the Security of Software Application by the Use of Fault Tree Analysis in Decision Making Special Issue of International Journal of Computer Applications on Advanced Computing and Communication Technologies for HPC Applications - ACCTHPCA, June 2012
8. Software Tool for Distributed Elevator Systems Scientific Publications of the State University of Novi pazar Ser. a: Appl. Math. Inform. and Mech. Vol. 3, 1 (2011).