International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 2; Issue: 3; March-2014

Change Requirement Traceability based Impact Analysis Methodology to evaluate Object-Oriented Software Systems

Sunil T D[1]
[1]Dept. of Electronics & Communication Engg.,
Sri Siddhartha Institute of Technology
Tumkur, Karnataka, India
[1]sunil.tumkur@gmail.com

Dr M.Z.Kurian[2]
[2]Dept. of Electronics & Communication Engg.,
Sri Siddhartha Institute of Technology
Tumkur, Karnataka, India
[2]mzkurianvc@yahoo.com

ABSTRACT

It is a well known fact that software maintenance plays a major role and finds importance in software development life cycle. As object-oriented programming has become the standard, it is very important to understand the problems of maintaining object-oriented software systems. This paper aims at evaluating object-oriented software system through change requirement traceability – based impact analysis methodology. The major issues have been related to change impact algorithms and inheritance of functionality.

Keywords

Change Requirement Traceability, Impact Analysis, Object-Oriented Software Systems, Software Maintenance, Change Impact algorithms, inheritance of functionality.

1. INTRODUCTION

There are several standards for traceability, such as ISO15504 and CMMI, Over the past decades, several techniques were developed for tracing requirements. Most of the traditional techniques like Trace-based Impact Analysis Methodology (TIAM), which is based on utilizing the trace information and Work Product Model (WoRM) , which is to define requirement change impact metric for determining severity in change requirements. The above methodology has predictive value for finding classes of similar changes. TIAM which is intended for planning rather than ensuring that changes are thoroughly implemented. TIAM potentially could be used to evaluate the risk of volatile requirements. In case of design changes, there are cognitive consequences of the object oriented approach. Novice designers have been found to have problems with class creation and articulating the declarative and procedural aspects of the solution. Accordingly, here it is to introduce traceability patterns or methods as a solution to requirement-component that can be applied to both traditional and modern development processes. This approach has achieved as a result of the conformance of the structure of the source code to the traceability patterns or methods. In the software life cycle, software undergoes changes at all stages. A software product is successful if a software changes are identified or managed from all the phases of software life cycle, like requirement specification phase, design phase, implementation phase and maintenance phase.

To obtain a software product it should be clear to have a well established threshold and it must get higher and higher as development proceeds or no product ever appears. Software maintenance consumes approximately forty percent of the software expenditure, since it is a non-trivial phase in software development lifecycle capturing traceability link between code and element in artifacts can be helpful in many tasks. Program comprehension, maintenance, requirement tracing, impact analysis and reuse of existing software. Many number of traceability patterns or methods were introduced to trace back elements from source code in reverse engineering. Traceability matrix, keywords, aspect weaving, information retrieval, scenario-based, event based, process centered, design pattern, goal centric and few examples of traceability methods. The demand to reengineer legacy system has increased significantly with the shift toward web-based user interface. The traceability patterns or methods are used for many reasons, such as managing evolutionary software changes, impact analysis, software architecture. The object-oriented paradigms such as classes and its relationship namely association, aggregations, dependencies, multiplicity have been conducted by many researcher. The objective of this paper is to create and provide round-trip engineering capability during traceability process.

Organisation: The literature survey about the related topic is dealt in section 2. Section 3 deals with the types of Traceability models. Impact analysis based on Change requirement traceability is discussed in section 4. The research results are presented in section 5. The paper is concluded mentioning the conclusive remarks in section 6.

2. LITERATURE SURVEY

There are a number of phases in the life of a software product. The waterfall model, as described by Ghezzi et al., [1], has five major phases. They are requirements analysis and specifications, coding and module testing, integration and system testing and delivery and maintenance. This research is concerned only with the final aspect of the final phase, maintenance. The maintenance phase is the longest phase of the life cycle. Maintaining software becomes more difficult as time progresses and the system evolves. Chandra Shrivastava et al., [2] stated that the algorithms calculate the transitive closure of each of the potentially effected classes

and methods. It will be possible to greatly improve upon the information provided by the algorithms in recognition of low-level design patterns, effects of data type changes, and effects of addition and deletion of classes can be drawn from the LLSA model of an object-oriented system.

Chen. X., Tsai et al., [3] presented an integrated environment for C++ program maintenance which describes three new dependence graphs specific to object-oriented software systems: message, class and declaration dependence in a model called C++ DG. Additionally, several new slicing techniques are presented. The use of the new dependencies and slicing on code maintenance is described. The dependencies are described, specifically as to the ripple effect and regression testing. The application of the discovered dependencies and program slicing leads to recursive analysis of the ripple effect caused by code modification. As the effects are located, classes and methods affected can be "marked" for testing or re-execution in the testing phase.

Li.,L et al., [4] explained four algorithms that measure the effect of proposed changes to object-oriented systems. The ripple effect is calculated by application of algorithms that

1. calculate the change effects inside of a class
2. calculate the change effects among clients
3. calculate the change effects among subclasses
4. measure the total effect by driving the algorithms in 1,2 and 3

The author also presented the details of how different types of changes affect the system. Changes are broadly categorized as method or member change, and then refined to more detail such as adding a member or changing an attribute. Gallangher, K [5] described about program slicing to select a point in an ANSI C program for observation. The method looks at program variables and essentially models dependencies that exist among variables via assignment statements and parameter passing. The method is a visualization of the data collected by the Surgeon's Assistant and is called the Decomposition Slice Display System. According to Hutchins et al., [6] Visual Impact Analysis has improved the recognition of further dependencies such as interference. Bohner.S.A [7] presented that software engineering practice evolves to respond to demands for distributed applications on heterogeneous platforms; software change is increasingly influenced by middleware and components. Interoperability dependency relationships now point to more relevant impacts of software change and necessarily drive the analysis. Software changes to software systems that incorporate middleware components like Web services expose these systems and the organizations they serve to unforeseen ripple effects that frequently result in failures. Current software change impact analysis models have not adequately addressed this trend. Moreover, as software systems grow in size and complexity, the dependency webs of information extend beyond most software engineers' ability to comprehend them. This paper examines preliminary research for extending current software change impact analysis to incorporate interoperability dependency relationships for addressing distributed applications and explores three dimensional (3D) visualization techniques for more effective navigation of software changes. Pressman [8] explained that as software system becomes larger and more complex, numerous corrections, extensions and adaptations tend to be more chaotic and unmanageable. The traditional way of addressing the maintenance task individually is no longer practical. It needs a special management system, called the Software Configuration Management (SCM) that covers the procedures, rules, policies and methods to handle the software evolution (IEEE, 1998b). SCM has been identified as a major part of a well defined software development and maintenance task. SCM deals with controlling the evolution of complex software systems that supports version controls and administrative aspects such as to handle change requests, and to perform changes in a controlled manner by introducing well-defined processes. Suhaimi Bin Ibrahim [9] illustrates that most of the Computer Aided Software Engineering (CASE) tools and applications focuses on the high level software and yet are directly applicable to software development rather than maintenance. While the low level software, e.g. code is given less priority and very often left to users to decide. This makes the software change impact analysis extremely difficult to manage at both levels. Secondly, there exists some research works on change impact analysis but the majority confine their solution at the limited space i.e. code, although more evolvable software can be achieved at the meta model level. No proper visibility is being made by the ripple effects of a proposed change across different levels of work product. If this can be achieved, a more concrete estimation can be predicted that can support change decision, cost estimation and schedule plan. M.Z.Kurian et al., [10] explained a comparative software maintenance methodology to assist in Object Oriented systems was carried out with main intention regarding to impact analysis and ripple effect to retesting of affected and changed components. This reduces the cost of testing and assists in identifying change impact in object-oriented maintenance. Since, it does not emphasize on the change requirement analysis and tracing object oriented software system it is to look forward with other methods. Ali R. Sharafat et al., [11] proposed an estimation of change-proneness of parts of a software system is an active topic in the area of software engineering. Such estimates can be used to predict changes to different classes of a system from one release to the next. They can also be used to estimate and possibly reduce the effort required during the development and maintenance phase by balancing the amount of developers' time assigned to each part of a software system. This is a novel approach to predict changes in an object-oriented software system. The rationale behind this approach is that in a well-designed software system, feature enhancement or corrective maintenance should affect a limited amount of existing code. The goal is to quantify this aspect of quality by assessing the probability that each class will change in a future generation. Peter Zielczynski [12] explained an approach which is applied to software writing in an object-oriented language to trace object oriented code into functional requirements. Here, it is addressed the problem of establishing traceability links between the free text documentation associated with development and maintenance cycle of a software system and its code. Further, vector space models to compare different model and to assess the relative influence of affecting factors are not considered.

In this paper, based on the requirement management to maintenance is considered so that change requirement traceability analysis is done on the requirement as well as object-oriented software systems and a round-trip traceability analysis is performed.

3. TRACEABILITY MODELS

Requirement traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backward direction. Forward traceability is the ability to trace

a requirement to components of a design or implementation. Backward traceability is the ability to trace a requirement to its source that is, to a person, institution, law, argument etc. Inter-requirements traceability refers to the relationships between requirements. Inter-requirement traceability is important for requirement analysis and to deal with requirements change and evolution Francisco A et al., [13]. Extra-requirements traceability refers to the relationships between requirements and other artifacts.

4.CHANGE REQUIREMENT TRACEABILITY BASED IMPACT ANALYSIS

It is the result of the elicitation process Gotel O.C.Z et al., [14]. The tracing of a requirement can be done in either way, to get information related to the process of elicitation, prior to its inclusion in the requirements specification or to get information related to its use, after the requirement has been elicited and included in the requirement. It has pre-requirements specification traceability and post-requirement traceability specification traceability. Pre- requirement traceability refers to those aspects of a requirements life prior to its inclusion in the requirement specification. Post RS-traceability refers to those aspects of a requirements life that result from inclusion in the requirement specification. Pre RS-traceability is used, when there is a change to a requirement and when to get the requirements source or people supporting it to validate change. Post RS traceability is used to get the design module to which a requirement was allocated or the test procedures created to verify the requirements.

Change Requirement Traceability Based Impact Analysis is a Non-functional tracing and Informal tracing that is, in functional tracing, those related to well establish mapping between objects model types and mapping types which allow analysis models, design models, process models, organizational models. The Non-functional tracing is related to the tracing of non-functional aspects of software development. They are usually related to quality aspects and results from relationships to non-tangible concepts. The traces that related requirements to goals, objectives, intensities and decisions are example of non-functional tracing. Non-function tracing are classified into four categories like reason, context, decision and technical.

The tracing of non-functional aspects of software development can be automatically performed only using a representation of that aspect. Therefore, here it is to use some model to functionally capture the non-functional aspects we want to trace, it may use an organizational model to relate policies, goal and roles to requirements, or it may be used process model to relate requirements to activities and resources. It is also an informal need for trace definition. The definition of traces and traceable objects should promote their uniform understanding. Differences and interpretation are the causes of errors, and in the more serious cases once may end up tracing what did not happen. To account for non-functional traces, the definition of traceable objects should allow the use of hyper-media objects like videos, recording and images together with mechanism for inspecting these kinds of objects. The relationship between recorded real world observations and parts of conceptual model is called extended traceability Haumer P et al., [15] Smith t et al., [16] Yu W.D. [17]. Sarah Maadawy et al., [18] presents a methodology to measure software complexity for changes. It studies attributes that affect complexity of change and the relation between requirements and each other to finally find a complexity measure the will serve in finding a precise estimate for the change. However, it did not discuss the object-oriented analysis and design aspects.

In this paper, the change requirement traceability based impact analysis methodology has been discussed, which is for a non-functional and informal and extended traceability, also object-oriented analysis and design aspects are discussed. This paper discusses the following phase's i.e.

Phase One

A. Validating the new requirements from any of the stake holders.
B. Classification of requirement whether functional or non-functional requirement
C. Traceability matrix can help tracing the requirement
D. Review of the Requirements
E. Requirement Evaluation
F. Requirement Documentation
G. Acceptance Testing

Phase Two

A. Stability: Unstable Requirements
B. Completeness: Incomplete Requirements
C. Clarity: Unclear Requirements
D. Validity: Invalid requirements
E. Feasibility: Infeasible requirements
F. Precedent: Unprecedented Requirements

Stability:

This represents the system vulnerability to change. It has been noticed that software maintainability degrades as changes are made to it which increases complexity of the software, system stability will be calculated as in

S(#NORS + #NOCNR + #NOCUR + #NOCDR) / (#NORS)

Where S= Stability and
NORS = No. of original requirement in the system
NOCNR=No. of cumulative number of requirement
NOCUR = No. of cumulative number number of requests updated in the system
NOCDR=No. of cumulative number of request deleted from the system.

Completeness

This represents completeness of the requirement

CMP=NARS –N IR
CMP=Completeness of the system
NARS=No. of Actual / original requirement in the system
NIR =Number of Incomplete Requirement in the system

Clarity

This represents clarity of the system.

CL=NARS –N IR-UCLR
CL=Clarity of the system
NARS=No. of Actual / original requirement in the system
NIR =Number of Incomplete Requirement in the system
UCLR=No. of Unclear requirements

Feasibility

This represents feasibility of the system.

FR=IFR -UCLR
FR=Feasibility requirements of the system
IFR =Number of Infeasible Requirement in the system
UCLR=No. of Unclear requirements

Precedent

This represents precedent of the system.

PR=CMP+CL+FR
PR=Precedent requirements of the system
CMP =Completeness of the system
CL=Clarity of the system
FR=Feasibility of the system

5.RESULTS

Case Study: Flight Booking System

Here it is to identify, visualize and analyze the change requirement traceability analysis on object-oriented software system. Here, Flight Booking System case study has been taken as a requirement. Based on the requirement level in, it is to split requirement into different requirement types

1. Stakeholders need
2. Feature
3. Use Case
4. Supplementary Requirement
5. Test Cases
6. Scenarios

1.The requirements at the top level of the levels (stakeholders' requests) are gathered using various methods of knowledge elicitation:

- Interviews
- Questionnaires
- Workshops
- Storyboards
- Role playing
- Brainstorming sessions
- Prototyping
- Use cases
- Analysis of existing documents
- Observation, task demonstration
- Analysis of existing systems

2. A business analyst derives the second level of the levels (features) from stakeholders' requests by cleaning the requirements and translating them from the problem domain to the solution domain. The features should have all the attributes of a good requirement:

- Unambiguous
- Testable (verifiable)
- Clear (concise, terse, simple, precise)
- Correct
- Understandable
- Feasible (realistic, possible)
- Independent
- Atomic
- Necessary
- Implementation-free (abstract)
- Consistent
- Non-redundant
- Complete

To fix the requirements that are missing at least one of these attributes, which can apply some of the following transformations:

- Copy
- Split
- Clarification
- Qualification
- Combination
- Generalization
- Cancellation
- Completion
- Correction
- Unification
- Adding details

3. The third layer of the levels contains use cases and supplementary requirements. Use cases capture functional requirements. Creation of use cases consists of the following steps:

1. Identify actors.
2. Identify use cases.
3. Design the initial use case model.
4. Structure the model.
5. Create use case documents.

4. Supplementary requirements capture mostly nonfunctional requirements. They may also capture some generic functional requirements not associated with any specific use cases. Supplementary requirements can be classified as follows:

- Functionality
- Usability (accessibility, aesthetics, user interface consistency, ergonomics, ease of use)
- Reliability (availability, robustness, accuracy, recoverability, fault tolerance, safety, security, correctness)
- Performance (throughput, response time, recovery time, startup/shutdown time, capacity, utilization of resources)
- Supportability (testability, adaptability, maintainability, compatibility, configurability, Upgradeability, install ability, scalability, portability, reusability, interoperability, Compliance, replace ability, changeability, analyzability, audit ability, localizability)
- Design constraints
- Implementation requirements
- Interface requirements
- Physical requirements
- Documentation requirements
- Licensing and legal requirements

5. Test cases are created to test the requirements from the third level. The following steps are used to derive test cases from use cases:

- Create scenarios.
- Identify variables for each use case step
- Identify significantly different options for each variable
- Combine options to be tested into test cases
- Assign values to variables

6. To create test cases from supplementary requirements, you can use one of the following approaches:

- Execute selected functional test cases in different environments
- Add checks to all use cases
- Check and modify a specific use case
- Perform the exercise
- Checklist
- Analysis
- White-box testing
- Automated testing

7. Design diagrams are also derived from the requirements on the third level, especially Use cases. Here are the possible approaches:
• Design classes that will capture required data and functionality
• Create one sequence diagram for each scenario
• Simultaneously add required methods and attributes to the classes on the class Diagrams

8. Documentation is created from various elements of the levels.

Algorithms for 'Book a Flight'

Step 1: Begin Algorithm
Step 2: Enter URL
Step 3: Enter flight data search flights
Step 4: Select a flight
Step 5: System Display return flights
Step 6: System Display details of flights
Step 7: Confirm the flight
Step 8: New User Register
Step 9: Login
Step 10: Provide passenger information
Step 11: Display available seats
Step 12: Select Seats
Step 13: Enter Billing information
Step 14 : Provide confirmation number
Step 15: End algorithm

Use Cases

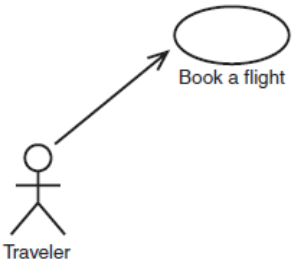

Figure 1.1 An ACTOR and a use case

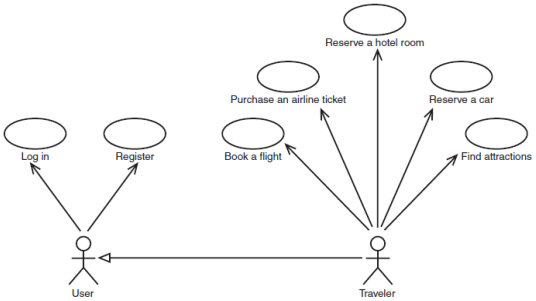

Figure 1.2 Use case Initiated by Travelers and User

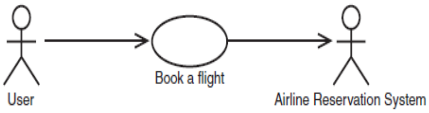

Figure 1.3 A Context diagram for the Use case book a flight

Traceability Structure

Figure 1.4 shows traceability structure in this case study

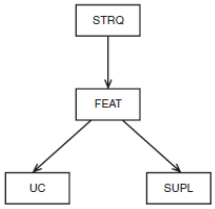

Figure 1.4 Traceability structure for case study "Book a Flight"

• Stakeholder Requests (STRQ) will be traced to Features (FEAT) defined in the Vision document and supplementary Requirements defined in the Supplementary Specification. There may be a many-to-many relationship between STRQ and FEAT, but usually it is one Stakeholder Request to many Features. Every approved Request must trace to at least one Feature or Supplementary Requirement.

• Feature Requirements (FEAT) (defined in the Vision document) will be traced to either a Use Case or Supplementary Requirement. Every approved feature must trace to at least one Use Case or Supplementary Requirement. There may be many-to-many relationships between Features and Use Cases and Supplementary Requirements.

• Use Case Requirements (UC) defined in the Use Case Specifications will be traced back to Features.

• Supplementary Requirements (SUPL) will be traced back to Features.

Object Oriented System Design from Use Cases

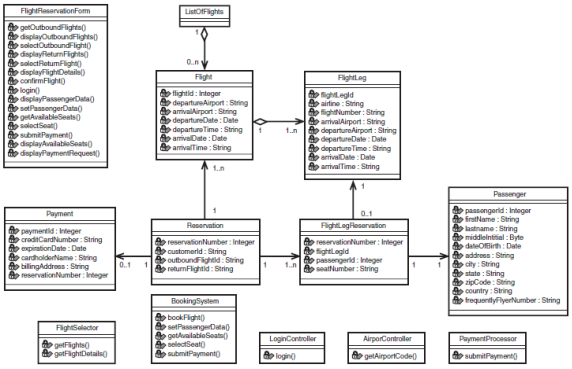

Figure 1.5 A class diagram showing classes that implement the functionality of a basic flow of the Book a flight use case.

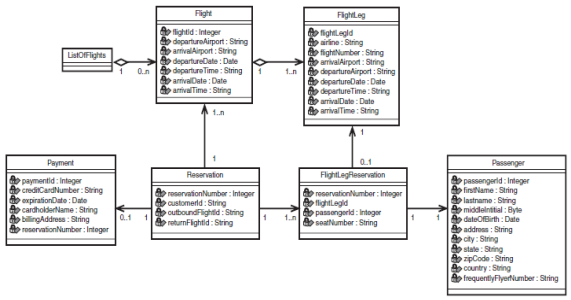

Figure 1.6 Class Reservation and related classes

6.CONCLUSION

The Change requirement traceability of a case study "book a flight" requirement provides an object oriented approach architecture till a class diagram. The object oriented analysis has been done for the case study, "book a flight" and arrived at a class diagram by using 'use cases' and arrived at an algorithm for "book a flight' case study. The proposed analysis is highly dependent on a very well defined software requirements specification and non-functional traceable requirement. Further based on change in the requirements the impact on the class diagram till test case attributes can be identified. Based on the requirement traceability which test cases must be changed can be identified and also impact of object-oriented paradigms can be analyzed.

7. REFERENCES

[1]. Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli, (1991), Fundamentals of Software Engineering, Prentice Hall Publishing.

[2]. Chandra Shrivastava, D. L. Carver, "Using Low-Level Software Architecture for Software Maintenance of Object-Oriented Systems", Proceedings of the 1995 Software Engineering Forum, Boca Raton, FL, November (1995), pp. 31-40.

[3]. Chen. X., Tsai. W., Hunag. H., Poonawala. M., Rayadurgam. S., Wang. Y., (1996), Omega-an Integrated Environment for C++ Program Maintenance, Proceedings of the International conference on software Maintenance, pp.114-123.

[4]. Li.,L.,Offutt,A.J., (1996), Algorithmic Analysis of the Impact of Changes to Object-oriented Software, Proceedings of the International Conference on Software Maintenance, pp. 171-184.

[5]. Gallangher, K., (1996), Visual Impact Analysis, Proceedings of the International Conference on Software Maintenance, pp. 52-58.

[6]. Hutchins,M., Gallagher,K., (1996) Improving Visual Impact Analysis, Proceedings of the International Conference on Software Maintenance, pp.294-301.

[7]. Bohner.S.A.,(2002). Software change impacts–an evolving perspective, Proceedings of the International Conference on Software maintenance, pp 263 – 272.

[8]. Pressman. (2004) A Dynamic Analysis Approach Concept Location. Technical report of Software Engineering.

[9]. Suhaimi Bin Ibrahim (2006), A Document-Based Software Traceability to Support Change Impact Analysis of Object-Oriented Software, University Teknologi Malaysia, Thesis, pp. 45-56.

[10].M.Z.Kurian and A S Manjunath (2007), Requirement traceability and impact analysis methodology to evaluate software requirements changes, National Conference on Trends in Advanced Computing, at DMCE,Airoli,Navi Mumbai,28-29

[11].Ali R. Sharafat and Ladan Tahvildari (2008), Change Prediction in Object- Oriented Software Systems: A Probabilistic Approach, Journal of Software, Vol. 3, No. 5, pp.10-38.

[12].Peter Zielczynski, (2013) IBM, Requirements Manangement Using IBM Rational Requisite Pro.

[13].Francisco A C Pincher, Requirement traceability Technical Report, University of Brasilia, (2000)

[14]. Gotel O.C.Z and Finkelstein ACW. An analysis of the requirements traceability problem. Proceedings of ICRE94, 1st Internation conference on requirements engineering, 1994, Colorado springs Co, IEEE CS Press (1994).

[15].Haumer P ., Pohl K., Weidenhaupt K and Jarke M . Improving reviews by extended traceability. Proceedings of 32nd Hawaii International Conference on system sciences volume 3; (1999), January 05-08; Maui, Hawaii.

[16]. Smith t,T J READS: A requirements engineering tool. Proceedings of RE'93, International Symposium on Requirements Engineering; 1993, January 4-6; san Diego,C.A. Los Alamitos,CA,IEEE Computer Society,(1993).

[17].Yu W.D. Verifying software requirements – a requirement tracing methodology and its software tool – RADIX, IEEE Journal on Selected Areas of Communication (1994);12(2):234-240.

.

[18]. Sarah Maadawy and Akram Salah, Measuring Change Complexity from Requirements: A proposed methodology, IMACST Volume 3, Number ,Feburary (2012).

Asst. Prof. Sunil T D received Bachelor Degree from Bangalore University in Electronics and Post graduate degree in Electronics from Visvesvaraya Technological University at BMSCE, Bangalore and Pursuing Ph.D degree in Software Engineering from VTU, Belgaum, Karnataka, India. Having 12 Years of Teaching experience in the field of Electronics & Communication Engineering. Published several papers in peer reviewed international journals, and several conference papers.

Dr M.Z.Kurian received his Bachelor Degree from Bangalore University and Post graduate degree in Industrial Electronics from Mysore University, and Ph.D degree in Software Engineering from Dr.MGR University, Chennai, Tamil Nadu, India. He has more than 30 Years of Teaching experience in the field of Electronics & Communication Engineering. Published several papers in peer reviewed international journals including IEEE, and several conference papers.