ISSN:1991-8178

Australian Journal of Basic and Applied Sciences

Journal home page: www.ajbasweb.com

A Survey of Fault Detection Mechanism

[1]Ms. Uma Rani and [2]Dr. T. Venkatachalam

[1]*Assistant professor, Department of Computer Technology and applications, Coimbatore Institute of Technology, Tamil Nadu, India.*
[2]*Professor, Department of Physics, Coimbatore Institute of Technology, Tamil Nadu, India.*

ARTICLE INFO

Article history:
Received 20 January 2015
Accepted 02 April 2015
Published 20 May 2015

Keywords:
Survey, UML, Model Based, Scenario Based, path oriented, goal-oriented and genetic approaches, Automatic Generation of Test cases.

ABSTRACT

Software Fault Detection is a process of fault recognition. It is achieved by combination of testing, debugging and monitoring. This paper presents a survey of various fault detection mechanism and the areas that needed improvement and futuristic research are presented.

To Cite This Article: Ms. Uma Rani and Dr. T. Venkatachalam., A Survey of Fault Detection Mechanism. ***Aust. J. Basic & Appl. Sci., 9(16): 51-56, 2015***

INTRODUCTION

First, let's understand the difference between Fault, Error and Failure.

Fault. is a physical defect, imperfection or flaw that occurs in hardware or software; it can also be defined as the mechanical or algorithmic cause of an error. E.g. infinite program loop, Algorithmic Faults like Missing initialization, Incorrect branching condition.

Error. is a deviation from correctness or accuracy and it is usually associated with incorrect values in the system state.

Failure. is a non-performance of some action that is due or expected, a system is said to have a failure if the service it delivers to the user deviates from compliance with the system specification. Generally in software terms the above can be summarized as Bug in a program is a fault. Possible incorrect values caused by this bug are an error. Possible crash of the operating system is a failure.

Software fault detection. by testing is an activity performed by means of generating test cases and the effectiveness of the test cases is determined by its ability to detect fault, error and failure. system knowledge and application skills to apply these techniques are very important for effective testing.

Debugging. is a support means to clear the failures.

The various testing activities followed in software field are Acceptance Testing, Unit Testing (Static Testing, Dynamic Testing like white box, black box, grey box testing), Integration Testing, System Testing, Regression Testing, Conformance Testing, Load Testing, Ageing Test, Compatibility Testing, Sanity Testing, Smoke Testing

II. Procedure for Studies Selection:

The methodological approach used for identifying the papers relevant for this study makes use of both an ad-hoc (manual) literature review, and a systematic (automatic) literature review. We used both manual and automatic searches as a way to maximize the retrieval of relevant papers (as advocated in e.g., (Prasanna, M. and K.R. Chandran, 2009)).

First, we formulated some research questions. Then, in the ad-hoc review, the author (who have different backgrounds and specialized knowledge in software architecture and engineering, respectively) selected relevant papers by looking at their own communities and networks and, more generally, at software engineering/architecture and dependability conferences and events. The selection has been carried out according to inclusion and exclusion criteria defined and finalized during the ad-hoc review. Conversely, the systematic literature review (SLR) was carried out to analyze, in a more systematic way, the body of knowledge in the field. The SLR uses the set of papers produced through the ad-hoc review as a pilot for testing the correctness of the SLR protocol.

A total of 50 papers have been selected and surveyed in this study, 27 coming from the ad-hoc review, 23 more from the SLR. The rest of this section formulates the research questions, defines

Corresponding Author: Ms. Uma Rani, Assistant professor, Department of Computer Technology and applications, Coimbatore Institute of Technology, Tamil Nadu, India.
E-mail: umarani79@yahoo.com

Australian Journal of Basic and Applied Sciences, 9(16) Special 2015, Pages: 51-56

how the ad-hoc study has been performed, and describes the SLR protocol used to refine the initial set of papers.

A. Research Question Formulation:

This study focuses on papers on fault detection at all the above introduced levels, which involves approaches to and solutions for detecting those faults (errors, exceptions, etc.) that are caused by failing components, connectors and other architectural elements, and affect the whole system.

In order to specify the aim of our research, a research question (with sub-questions) were formulated, as described below: Research Question: What are the approaches to architecting Fault detection?

_ Sub-question: What are the methods, and processes leading to Fault detection?

_ Sub-question: Which style/pattern is more appropriate for Fault detection?

They are used to drive the papers selection process.

B. Ad-hoc Study:

In the ad-hoc review, the paper selection process was driven by the authors' experience of the field and web based searches using the most relevant search engines. We searched into the main conferences and journals on Software Engineering (ICSE - Int. Conference on Software Engineering, ESEC/FSE

- European Software Engineering and Foundations of Software Engineering Conference, FSE - Foundation of Software Engineering, IEEE TSE

- Transactions on SE, ACM TOSEM - Transactions on SE and Methodology, JSS - Journal on Software and Systems), CBSE

- Int. Symposium on Component-based Software Engineering, to supplement our findings, we ran searches on Google and Google Scholar.

As a way to drive the selection process, we incrementally defined inclusion and exclusion criteria, according to the research questions.

To be included, a paper needs to propose novel principles, elements and styles, development processes and specification languages for reasoning about fault detection and its specific steps at all the developmental stages. These would allow developers to explicitly represent abnormal system behavior and state. Papers not validated by the computer science community are excluded: only peer-reviewed sources are included, while technical reports and PhD thesis are not. This is consistent with our overall aim of analyzing fault detection approaches which are designed to make applications in good working condition. Papers which do not explicitly mention fault detection, automatic test case generation, model based etc., in general, or its parts are excluded.

C. The Systematic Literature Review:

The Systematic Literature Review (SLR) is a methodology created to standardize the methods of analyzing research in particular scientific fields. Kitchenham (Ghosh, S., 2003) defines the SLR as"a means of evaluating and interpreting all available research relevant to a particular research question, topic area or phenomenon of interest". The SLR can use interpretive synthesis, which aims to obtain specific-context knowledge through research articles without any background or initial data-set, or an integrative review, in which the reviewing process is based on a background of knowledge in the area of interest, which makes it possible to draw up an initial list of (pilot) papers. In this work, we use the initial pool of papers selected in the ad-hoc study to run an integrative review process.

The systematic review process includes a number of steps. Following we defined our SLR protocol by defining a suitable search string (used to match the set of research questions in the best possible way), a time period (adequate to capture as many relevant papers as possible), a list of web search engines (used to automatically identify a list of papers, potentially related to the review research topic), and a studies selection process (used to select those articles that better fits our research questions).

Search String:

Based on the study research questions, we identify some relevant keywords and combined them in order to form the study research question. The search string we identified and used in the SLR is: ("Survey") AND ("fault detection" OR "Testing" OR "Automatic Test Case Generation"). Although other relevant keywords had been identified, they were discarded in order to have a query that would be both strict and as complete as possible.

Time Period:

In order to minimize the risk of missing relevant papers in the area, we took into account all articles published from January 1, 1995 to July 1, 2013. This time period was carefully selected in order to cover as many relevant publications as possible.

Web Search Engines:

The search string was run in the following search engines: ACM the Guide to Computing Literature, ISI Web of Science, and Springer Link. The ACM - The Guide search engine includes all the articles retrievable from ACM Portal Digital Library, IEEE Computer Society, and Elsevier.

III. Literature Survey:

Many approaches have been already presented and with the advent of modeling tools many different test cases based on the details of the specifications, the design models, the data mining models, the Regression Models have been proposed. The approaches for fault detection means have been classified on these different categories.

Australian Journal of Basic and Applied Sciences, 9(16) Special 2015, Pages: 51-56

IV. Specification Based Test Case Generation:

This is based mainly on combining the Functional Testing [Black Box] and Structural Testing [White box]. This category mainly uses the Functional Specification to identify the Independently testable feature which is used to identify the representatives values [representatives of equivalence classes that are apt to fail often or not at all].Then the representatives values are used to derive the Test case specification through which the test case gets generated.

Functional Testing [Black Box]:

1. Boundary value testing involves max, min values, robustness, generalized and worst case testing.
2. Equivalence class testing
3. Decision table-based testing
4. Combinatorial testing involves search-based and algebraic methods.

Structural Testing [White Box]:

1. Path coverage criteria:

A set of concurrent message paths P satisfies the all-message-paths coverage criterion if and only if P contains all start-to-end message paths in a sequence diagram (Offutt, A.J. and A. Abdurazik, 1999). A start-to-end message path in a sequence diagram is a sequence of messages that begins with an externally generated event and ends with the production of a response that satisfies this event.

The primary test coverage criterion for interaction diagrams was first identified by Abdurazik (2000), and a comprehensive work was done by Ghose *et al*. (2003). They identified four different criteria for testing interaction diagrams (1) condition coverage criterion, (2) full predicate coverage criterion, (3) each message in link criterion, and (4) all message path criterions. The work in (Winters, M., 1999) identifies coverage criteria for sequence diagrams. But testing each message in a SD is not feasible since a system can have very high number of message links. Since the work done was pre-UML 2.0, it did not cover iterations, asynchronous messages and concurrent paths. The work presented in (Winters, M., 1999) also uses similar test adequacy criteria for testing sequence diagrams, but it overcomes the short comings of (Briand, L.C. and Y. Labiche, 2002; Ghosh, S., 2003) in addressing these issues. The testing approaches attempted in (Riebisch, M., 2003; Fröhlick, P. and J. Link, 2000; Tonella, P., A. Potrich, 2003) focus on the coverage of use case scenarios. These works can be considered essentially black box in nature and do not take into consideration the structural and behavioral aspects.

2. Logic coverage criteria:

A logical expression is a predicate, and the predicate consists of clauses, conjoined by Boolean operators (and, or,..). A clause contains no Boolean operators. Predicates and clauses occur everywhere, not only in source code. For example, test models often consist mainly of logical expressions. UML state charts, as another example, have predicates in terms of OCL expressions. Predicate coverage is the most basic logical coverage criterion, and there are usually many different ways to satisfy it. It is also known as Branch coverage, Decision coverage, Basic criterion. For each clause in a predicate: Evaluates to true, Evaluates to false Also known as: Condition coverage, Basic condition coverage.

Infeasibility:

- $(a > b \wedge b > c) \vee c > a$
- $(a > b) = true$, $(b > c) = true$, $(c > a) = true$ is infeasible
- Infeasible test requirements have to be recognized and ignored
- Recognizing infeasible test requirements is hard, and in general, undecidable
- More complex criteria also produce more infeasible test requirements

Generating tests for logical coverage criteria offers some challenges: First, we need the program to execute the path that leads to the predicate (reachability). Second, we need input values that indirectly assign values such that the clauses evaluate as needed. For example, for the clause a<b we need to find suitable values for a and b. This is explained through the triangle example.

3. Dataflow coverage criteria:

The main concept in data flow testing is a Definition Use pair. A definition use pair consists of a definition of a variable, a use of the same variable, and at least one definition-clear path from with the definition reaches the use without being redefined. Calculating DU pairs and Searching all paths is not feasible, Without loops, number of paths is exponential to number of nodes, with loops forget it, Aliasing of variable clauses serious problems!, Working things out by hand for anything but small methods is hopeless.

4. Mutation testing:

Original code with difference in syntax change is termed as mutation. Test cases are generated until the maximum mutants have been killed and their surviving is stopped. Automated-Generating Test Case Using UML State charts Diagrams by supaporn kansomkeat and wanchai rivepiboon experimented on the automatic testing technique to solve partially the testing process. This technique can automatically generate and select test cases from UML state chart diagrams. Firstly, transform this diagram into intermediate diagram, called Testing Flow Graph (TFG), explicitly identify flows of UML state chart diagrams and enhance for testing. Secondly, from TFG generate test case using the testing criteria that

Australian Journal of Basic and Applied Sciences, 9(16) Special 2015, Pages: 51-56

is the coverage of the state and transition of diagrams. Finally, the evaluation is performed using mutation analysis to assess the fault revealing power of our test cases.

V. Model Based Test Case Generation:

In the field of software testing in path toward generating test cases most of them use modeling language to generate test case. Unified Modeling Language (UML) is an Object Management Group (OMG) Standard for object-oriented modeling that has gain widespread use in the software industry (OMG, 2011; OMG, 2004). UML diagrams have many diagrams like class, state, interaction, component, usecase, sequence, activity, deployment diagrams that is used to depict both the static and dynamic state of the system.

Even though model based test case derivation has been extensively discussed in the literature, work using UML models is scarce (Abdurazik, A., A.J. Offut, 2000; Basanieri, F., 2001; Briand, L.C. and Y. Labiche, 2002; Hartmann, J., 2005; Offutt, A.J. and A. Abdurazik, 1999; Emanuela, G., 2007). Offut *et al*. (1999) proposed the idea of utilizing UML state chart diagrams, to generate test cases for unit level testing. Abdurazik *et al*. (2000) have proposed a technique using the UML collaboration diagram for generating test cases. Briand *et al*. (2002) proposed a complete system testing methodology. They used the widely used approach of finding sequential dependencies between use cases as proposed in (Bruegge, B. and A.H. Dutoit, 2000) and the use case scenarios as basis of generating test sequences. These scenarios are then identified by interaction diagram (either a sequence diagram or a collaboration diagram). The work uses OCL to present the conditions, guards and messages. Also it makes some modifications to the sequence diagrams for ordering of messages. Our work bears some similarity with the work of (Briand, L.C. and Y. Labiche, 2002) on deriving operation sequences from use case diagrams. The main difference lies in the fact that the authors make use of path-wise regular expressions conditions from models with analysis document like class descriptions and data dictionary rather than full predicate coverage generated from UML models. Furthermore, they have used scenario coverage focusing more on use case diagram. Sharma *et al*. (2007) generated the test case using structural coverage like use case and message path criterion from sequence diagram for system testing.

The work reported by Samuel *et al*. to generate test cases using dynamic slicing criterion. They have considered only one scenario of one use case using UML 1.X sequence diagram. In comparison to the work we have generated test cases using dependency and synchronization among use cases through UML 2.0 sequence diagram. Lastly, some of the early approaches are interesting, but too general as they lack detailed, operational descriptions.

Although UML provides a powerful mechanism for describing software that is safety-critical or that must be highly reliable, very less work has been done in the area of utilizing these models for testing purpose. Santosh, Mohapatra and Rajib explains that the test cases are derived from analysis documents or the artifacts such as use cases, their corresponding sequence diagrams and constraints specified across all these artifacts. They clearly depict the concepts of UDG, CCFG from respective UML diagrams for test sequence generation. They have focused on testing the sequences of messages and their collaboration in the various designed scenarios. Their testing strategy is mainly aimed at the usage of coverage criteria. By the understanding of their proposal we are able to appreciate the importance of Automation and its importance in the generation and coverage of testing the functionalities. And also that manual test design is time consuming and error-prone. It is necessary to develop automatic tests design techniques. Their proposal can be targeted and used for the phases of integration and system testing accommodating the message sequence and dependency information associated with the functional scenarios. The coverage obtained helps in detecting race conditions and inter dependency of functionalities and interactions, the collaboration of various objects and their defects.

Vi. Data Mining Based Test Case Generation:

M. Prasanna and K.R. Chandran. suggested an approach for UML based behavioral identification aspect of any system and identifying functionalities on the basis of Genetic algorithm. Used to generate optimal test cases which also can be consider as data mining approach .Case study on object diagram for ATM application is illustrate. Automation of test cases using C++, Java languages has been proposed. The functionalities required for testing is derived by analyzing the runtime changes and impacts due to their system dependencies. They clearly explain the object UML diagrams. Their proposal is to clearly identify bring out all possible functionalities of a given UML diagram. They also depict the various fault levels at different phases of testing such as unit , integration and system. They also discuss on the evaluation and the effectiveness of their process by means of mutation testing, This proposal talks about faster identification of defects in various phases to enable quicker completion of the product.

Vii. Regression Based Test Case Generation:

The Regression testing is a used to find out the changes do not impact the original software behavior. The cost of executing the updated test suites increases exponentially as software gets updated or version gets changed because of new features or of maintenance proposition, this becomes very costly to perform the regression with the new impacted functionalities. Various approaches have

been proposed in the areas of new impacted features identification, the new features dependency with the existing features, unique methods of selection of those features and functionalities so as to reduce the cost of the regression. This process of reduction of the test cases by the above informed means helps to identify the unwanted functionalities which make the regression to run faster as the number of test cases and its dependencies are reduced. This above explained process helps in quicker and faster identification of faults there by increasing the regression coverage. The proposed article also discusses some of the limitation and directions for future improvements.

This also provides details of all the existing proposal and also their respective problems and solutions. Hence by this means the importance of regression is emphasized targeting the future needs

VIII. Conclusion:

Software fault detection, what constitutes fault detection and the means of fault detection using test case generation has been clearly captured in this survey paper apart from the same there are other trends that affect the fault detection mechanism. To highlight and summarize the few dependencies of methodology and practices that has been impacting the detection process in the industry are like

- Agile that focuses on constant changes and levels of uncertainty vs. traditional approach that aims to attain "process maturity".
- Exploratory approach that focuses on execution during creation time vs. scripted approach that believes on creating and designing tests before they are executed
- Manual approach believes test automation is costlier and not applicable to all systems whereas agile community believes in automation of all test cases.
- The main question of when the fault detection start s during the creation, design, implementation or on the entire whole process?

To conclude, on the basis of the surveyed literature, statistically 60-65% of software faults originate from incomplete, missing, inadequate, inconsistent, unclear requirements, 35-40% of software faults originate from coding mistakes and proportional to size of code and number of paths in code. To fulfill the above challenges that occur in testing we need to automate the testing since automation takes in to account of the ever changing needs of the industry with minimum efforts to get the desired output.

REFERENCES

A Survey of UML-Based automatic Test Case Generation for Software Testing. Shanthi, A.V.K., D. Parthiban, Dr.G. MohanKumar, 2012. IJCSA-2012.

Abdurazik, A., A.J. Offut, 2000. Using UML collaboration diagrams for static checking and test generation. In Proceeding of the 3rd International Conferece on the Unified Modelling language(UML), Lecture Notes in computer Science, 1939: 383-395, York, UK, Springer-Verlag, GmbH.

Advanced Coverage Criteria -Software Engineering: Gordon Fraser Saarland University.

Basanieri, F., A. Bertomated, E. Marchetti, A. Rinoline, G. Lombardi and G. Nucerga, 2001. An Automated Test Strategy Based on UML Diagrams. In Proceeding of the Ericsson Rational User Conference, Upplands Vasby Sweden, October 10-11.

Bertolino, A., F. Basanieri, 2000. A practical approach to UML-based derivation of integration tests, In Proceedings of the Fourth International Software Quality Week Europe and International Internet Quality Week Europe (QWE), Brussels, Belgium.

Binder, R.V., 1999. Testing Object-Oriented Systems Models, Patterns, and Tools. Object Technology Series. Addision Wesley, Reading, Massachusetts.

Booch, G., J. Rumbaugh and I. Jacobson, 1999. The Unified Modeling Language User Guide, Object Technology Series Addision Weslley.

Briand, L.C. and Y. Labiche, 2002. A UML-Based Approach to System Testing. In Proceeding of the 4th International Concepts, And Tools, Springer-Verlag, London, pp: 194-208, October 01-05.

Bruegge, B. and A.H. Dutoit, 2000. Object-Oriented Software Engineering-Conquering Complex and Challenging Systems, Prentice Hall.

Debasish Kundu, Debasis Samanta, 2009. A Novel Approach to Generate Test Cases from UML Activity Diagrams, in Journal of Object Technology, 8(3): 65 -83.

Design of Fault Tolerant Systems - Elena Dubrova, ESDlab.

Dustin, E., 2003. Effective Software Testing: 50 Specific Way to improve your Testing. Addision-Wesley.

Emanuela, G., Franciso and Patricia, 2007. Test Case Generation by means of UML sequence diagrams and Labeled Transition Systems, IEEE, pp: 1292-1297.

Eriksson, H., M. Penker, B. Lyons and D. Fado, 2003. UML 2 Toolkit, Wiley.

Fraikin, F., T. Leonhardt, 2002. SeDiTeC-testing based on sequence diagrams, In Proceedings of 17th IEEE International Conference on Automated Software Engineering, pp: 261-266.

France, R., S. Ghosh, T. Dinh-Trong and A. Solberg, 2006. Model-driven development using UML 2.0: promises and pitfalls, IEEE Computer, 39(2): 59-66.

Fröhlick, P. and J. Link, 2000. Automated Test cases Generation from Dynamic Models, In Proceedings of the European Conference on Object-Oriented Programming, Springer Verlag, LNCS 1850, pp: 472-491.

Garousi, V., L. Briand, Y. Labiche, Control flow analysis of UML 2.0 sequence diagrams, Technical report. Available at http://www.sce.carleton.ca/squall/pubs/tech_report/TR_S CE-05-09.pdf.

Ghosh, S., R. France, C. Braganza, N. Kawane, A. Andrews, O. Pilskalns, 2003. Test Adequacy Asessment for UML Design Model Testing. In Proceedings of 14th International Symposium in Software Reliability Engineering (ISSRE), pp: 332.

H. Zhang, M. A. Babar, and P. Tell, Identifying relevant studies in software engineering, Inf. Softw. Technol., 53(6): 625-637. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2010.12.010.

Hartmann, J., M. Vieira, H. Foster and A. Ruder, 2005. A UML-based Approach to System Testing, Journal of Innovations system Software Engineering, 1: 12-24.

Ince, D., 1991. Object-Oriented Software Engineering with C++, McGraw-Hill.

Kitchenham, 2007. Guidelines for performing systematic literature reviews in software engineering, Software Engineering Group, School of Computer Science and Mathematics, Keele University, and Department of Computer Science University of Durham Durham, UK, Tech.Rep. EBSE Technical Report EBSE-2007-01.

McQuillan, J.A. and J.F. Power, A survey Of UMl-based Coverage Criteria for Software Testing. A Technical Report. National University of Ireland.

Mohanpatra, D.P., R. Mall, R. Kumar, 2005. Computing dynamic slices of concurrent object oriented programs, Information and Software Technology, 47: 805-817.

Muchnick, S., 1997. Advanced Compiler Design and Implementation, First Ed., Morgan Kaufmann.

Myers, G.J., C. Sandler, T. Badgett and T.M. Thomas, 2004. The art of software Testing, 2nd Edition. Wiley.

Navarro, P.L., D.S. Ruiz and G.M. Perez, 2010. A Proposal for Automatic Testing of GUIs Based on Annotated Use Cases Advances in Software Engineering, Article ID 671284, doi: 10.1155/2010/671284.

Nayak and D. Samanta, 2010. Automatic Test Data Synthesis using UML Sequence Diagrams. Journal of Object Technology, 09(2): 75-104.

Offutt, A.J. and A. Abdurazik, 1999. Generating Tests from UML specifications, In Proceedings of the 2nd International Conference on the Unified Moderating Language (UML'99), pp: 416-429.

OMG, 2004. XML Metadata Interchange (XMI),v2.1.

OMG, 2011. Unified Modeling Language Specification (v2.4.1).

Philip Samuel, Rajib Mall, 2008. A Novel Test Case Design Technique Using Dynamic Slicing of UML Sequence Diagrams, e-Informatica: Software Engineering Journal, 2(1).

Prasanna, M. and K.R. Chandran, 2009. Using Genetic Algorithm, Int. J. Advance. Soft Comput. Appl., 1(1): 19-32.

Regression Testing Minimisation, Selection and Prioritisation ;A SurveyShin Yoo, Mark Harman

Regression Testing Minimisation, Selection and Prioritisation, 2011 A Survey Shin Yoo, Mark Harman.

Riebisch, M., I. Philippow and M. Gotze, 2003. UMLBased Statistical Test Case Generation, in the Proceeding of ECOOP 2003, Springer Verlag,LNCS2591, pp: 394-411.

Rountev, A., O. Volgin, M. Reddoch, 2004. Control flow analysis for reverse engineering of sequence diagrams. Technical Report OSU-CISRC-3/04 –TR 12, Ohio State University.

Rountev, A., S. Kagan and J. Sawin, 2005. Coverage Criteria for testing of object interactions in sequence diagrams, In Fundamental Approaches to Software Engineering, Edinburgh, Scotland, 2-10.

Santosh Kumar Swain(1), Durga Prasad Mohapatra(2) and Rajib Mall, 2010. Int.J. of Software Engineering, IJSE., 3(2).

Sarma, M., R. Mall, 2007. Automatic Test Case Generation from UML Models, 10th International Conference on Information Technology, IEEE computer society, pp: 196-201.

Software Testing Methods and Techniques Jovanović, Irena

Specification-based Testing - Software Engineering - Gordon Fraser, Saarland University.

Supaporn Kansomkeat and Sanchai Rivepiboon, Automated-Generation of Test Case Using UML State chart Diagrams, SAICSIT 2003.

Testing Object-Oriented Software Systems, 2012. A Survey of Steps and Challenges [International Journal of Computer Applications (0975 – 8887) Volume 42– No.8, March 2012] - Clarence J M, N Ganesan, Anupam Ghosh, Nirupam Ghosh.

Tonella, P., A. Potrich, 2003. Reverse Engineering of the Interaction Diagrams from C++ Code, In Proceedings of IEEE International Conference on Software Maintenance, pp: 159-168.

Winters, M., 1999. Quality Assurance for object-oriented Software- Requirements Engineering and Testing w.r.t. Requirements Specification(in German),Ph.D thesis, University of Hagen, Germany.