ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 2, Issue 5, May 2013*

# A study on program verification using EFSBI approach

P.Joshna[1] , E.Sandhya[2] , T.Lakshmi[3]

Dept of Information Technology, Sree Vidyaniketan Engineering College, Tirupati, Andhra Pradesh, India

**Abstract:** Testing plays an important role in any software industry. Now a day it is very difficult to release completely defect free product. For detecting defects external formal specification based inspection is used. It can be carried out by step by step process and include five activities. First step, functional scenarios are derived from specification. Second step is paths are derived from program. Third step is linking from scenarios to paths. Fourth step is to reading and analyzing of paths against the corresponding scenarios and finally inspection report is produced. For increasing the effectiveness of formal inspection methods it can be applied in SPRT (Specification-Based Program Review Tool). Finally to compare this method with the perspective based reading then the results shows that the method is less effective in implementation-related defects rather than functional-related defects.

**Keywords:** Formal methods, Static analysis, validation of programs.

## I. INTRODUCTION

Static analysis [1] is the analysis of computer software which doesn't actually require executing or running the software. Static analysis tools only look on non runtime environment such as coverity-static analysis tools. Coverity-static analysis [2] is best-in class analysis engine it identifies the most critical errors in C/C++, Java and C# code bases. In single analysis, it can be referred as hundreds of users, thousands of defects and million lines of code [3]. Static analysis techniques only give importance to administrative aspects such as meetings and managements [4] and also it is only concentrate on implementation-related defects rather than function-related defects.

For overcoming the above defects extended formal specification based inspection is used. The inspection objective is to determine whether every functional scenario derived from the specification is correctly implanted by a set of program paths of the program. The formal inspection method mainly concentrate on requirement related errors and function-related defects. By using specification language such as VDM, Z and SOFL (structured-object oriented formal language) [5] some sort of formal methods are derived. SOFL is a combination of structured language, formal language and object-oriented language. By using this code quality is improved, enhance efficiency and reduce human error during inspection and improve the process quality of software inspection.

Perspective-Based Reading (PBR) technique [6] used for software inspections from the Scenario-Based Reading (SBR) family of reading techniques designed for defect detection in a requirements document. The main aim of perspective-based reading gives developers a set of procedures to inspect software products for defects. For correcting and detecting these defects early in the development process can save a lot of time and money and possibly avoid some embarrassment. But it is not effective in detecting functional-related defects rather than in implementation-related defects. So that extended formal specification based inspection method [7] is used to cover the above mentioned defects.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 2, Issue 5, May 2013*

## II. PROCESS INSPECTION

An inspection is a visual examination of a software product to detect and identify software anomalies, including errors, defects and deviations from standards and specifications. The inspection method is supported by inspection process. The process consists of five activities: functional scenarios are derived from specification; paths are derived from program; linking scenarios to paths; analyzing paths against corresponding scenarios; and finally produce an inspection report. Every activity that describes an operation is represented by a diamond in the figure and every data item is represented as a "specification language" or "functional scenario", which is represented by a box. An arrow from box to an operation means that the data item of the box is an input to the operation. An arrow from one operation to another represents a control flow.

To derive functional scenarios activity takes a specification as input and transforms it into an FSF (functional scenario form) from which all the scenarios are obtained. The important idea of the transformation is first to convert the post condition into a disjunctive normal forms (DNF) using a standard algorithm and then transform it into an FSF. The transformation from DNF to FSF takes several steps. At the each disjunctive clause in the DNF is transformed into a conjunction of a guard condition and a defining condition. Second activity, all the disjunctive clauses with the same guard conditions are merged into a conjunction of the guard condition and the disjunction of their defining conditions. Third activity, the precondition and each merged disjunctive clause are conjoined to build scenarios.

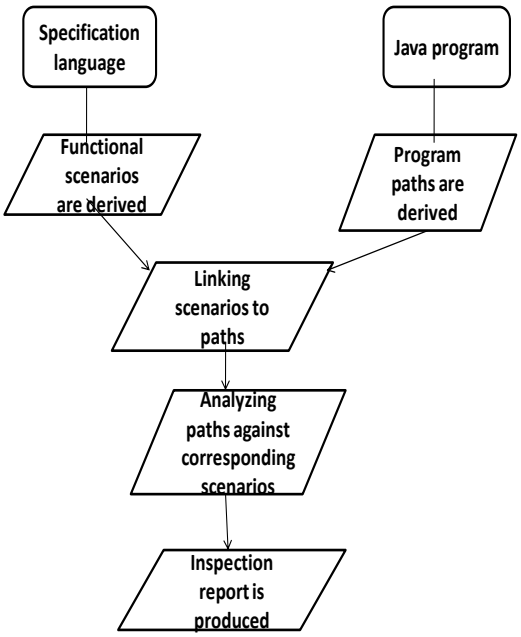


Fig: process for inspection

To derive program paths activity takes a program as input and derives all necessary program paths for analysis. To address this problem, we adopt the following strategy for expressing paths: For a sequential structure such as C1; C2, we produce an execution sequence denoted by [C1; C2], which is part of a related path. For if-then conditional structure such as if (e) and C, we produce two sequences [e; C] and [!e], respectively, where !e represents the negation of the logical expression e. For while loop such as while (e) C, we produce two sequences [while e, C,!e] and [!e] to represent all the possible sequences resulting from executions of the loop.

To Link scenarios to paths activity takes both derived functional scenarios and program paths as input to generate an inspection target lists (ITL) and a checklists. The checklist is related to the ITL: It contain a set of questions which can be derived automatically from the inspection target in the ITL. For example, we can derive the following questions from the target (f, q): "Is each symbol in f correctly implemented by q?", "Is each atomic condition in f correctly implemented by q?" so on . The

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 5, May 2013

above questions are carried out by a four level analysis are discussed in next activity.

To analyze paths against the corresponding functional scenarios step uses the checklists and the scenarios as guidelines to analyzing the related paths. Since the program paths are located in the program itself and analyzing them usually requires the inspector to refer the related contextual information (e.g., variable or operation declaration), the program itself also necessarily used for this step. For example, to answer the above mentioned questions derived from the target (f; q), the analysis can be done at four levels such as:

1. The symbol levels,
2. The atomic condition levels,
3. The condition levels, and
4. The scenario levels.

The four-level analysis follows the "divide-and-conquer" principle [8]: Check components and combinations.

At last activity, produce an inspection report, generates a document to report all the defects revealed during the analysis and to provide comments concerning any suspicious statements or conditions on the paths. The defects are detected need to be corrected by modification of program and the comments are served as a reminder messages for further clarification or confirmation by the individuals concerned (e.g., programmers, analysts, inspectors, or all of them).

## III. EXPERIMENTAL COMPARISON

Applying this approach, we have used 84 undergraduates to conduct an experiment on our FSBI comparing it with one of the popular inspection technique known as perspective-based reading [6]. Both methods are applied to part of a banking systems, and the results are analyzed and compared in three different ways: 1) how effectively FSBI works against the PBR in defect detection, 2) how the inspector is able to relate effective use of FSBI and PBR, respectively, and 3) what are the challenges that the inspectors are likely to face in using FSBI, and how they can be managed.

Perspective-Based Reading (PBR) is a technique for software inspection from the scenario-based reading (SBR) family of reading techniques are designed for defect detection in a requirement document [2]. The percentage to each average defects found denotes the average (defect) detection effectiveness for that category, which is calculated using the following formulas:

Average detection of effectiveness
= (average number of defects found / total number of defects) × 100%

Average number of defects found

$$= (d_1+d_2+\cdots\cdots\cdots+d_n) / n$$

## IV SPECIFICATION-BASED PROGRAM REVIEW TOOL (SPRT)

Effective tool supports are crucial for successfully applying software review techniques in practice. We describe the design and implementation of a software tools to support an approach to reviewing programs on the basis of their formal specifications. The approaches were initially proposed in the previous publications to improving the rigor, repeatability, and effectiveness of existing code review methods. The Specification-Based Program Review Tool (SPRT) [9] improves the effectiveness of our inspection method by taking the following steps: Functional scenarios are automatically generated; program paths are automatically generated, mapping scenarios to paths, support for reading and analysis of the code, supports for the input of defect descriptions and comments.

## V. RELATED WORK

To explain why formal specification techniques are used to discover problems in system requirements. To describing the use of algebraic techniques for interface specification and also describes the use of model-based techniques for behavioral specification The use of formal

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 2, Issue 5, May 2013*

specification to provide the inspection of programs was assessing safety critical software for the Darlington Nuclear Plant in Canada [10]. This technique known as Document Driven Inspection (DDI) [11] , was developed to cope with perceived difficulties experienced in the above project Formal specification techniques are part of a more general collection of techniques that are known as formal methods. These all are based on mathematical representations and analysis of software. Formal methods include formal specifications, specification analysis and proofs, transformational development, program verification. The principal benefit of formal method is in reducing the number of faults in systems. As a result, their main area of applicability is in critical system engineering. There are several successful projects where formal methods are used in this area. In this area formal method is used to be cost-effective because high system failure costs must be avoided. The formal Specification was written in Parnas' SCR (Software Cost Reduction) tabular notation [12] to explain the desired functions for the program, where the program is a module or a segment Although there is a relatively high cost in method development and education for the initial users of the method, the inspection discovered many unsuspected deficiencies between the code and the requirements. In the first activity, the functional table reflecting the behavior of the code is derived manually from the code and then compared with the tables in the design document. In the second activity, the function table in the design is compared with the tables in the requirement documents, with two preliminary goals: 1) prove that the behavior described in the design matches the requirements, and 2) identify behavior in the design that is specified in the requirements, and show that it is justified and that it cannot negatively affects the required behavior. By applying inspection methods, many defects are found in particular the "critical" ones, such as those which are causing runtime crashes, infinite loops, and unreachable code, may be detected and eliminated before testing begins, which will make testing more effective.

## VI. CONCLUSION

For verifying and validating the programs extended formal specification based inspection method is used. The main aim of this method is to use inspection to determine whether every functional scenarios defined in the specification is correctly implemented by a set of program paths of the program and whether every path is correctly implemented by a program. This method is more effective in detecting functional-related defects rather than implementation-related defects. By using specification based program review tool to improve the effectiveness of the inspection method.

## REFERENCES

[1] T. Gilb and D. Graham, "Software Inspection". Addison Wesley, 1993.
[2] "Coverity Static Analysis", http://www.coverity.com/products/ static-analysis.html, 2012.
[3] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. Mcpeak, and D. Engler, "A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World," Comm. ACM, vol. 53, no. 2, pp. 66-75, Feb. 2010.
[4] O. Laitenberger, "A Survey of Software Inspection Technologies," Handbook of Software Eng. and Knowledge Eng., pp. 517-556, World Scientific Publishing, 2002.
[5] S.Liu, "Formal Engineering for Industrial Software Development Using the SOFL Method". Springer-Verlag, 2004.
[6] V.R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, and M. Zelkowitz, "The Empirical Investigation of Perspective- Based Reading," Empirical Software Eng., vol. 2, no. 1, pp. 133-164, 1996.
[7] G. Babin and F. Lustman, "Application of Formal Methods to Scenario-Based Requirements Engineering," Int'l J. Computers and Applications, vol. 23, no. 3, pp. 141-151, 2001.
[8] D.L. Parnas and D.M. Weiss, "Active Design Reviews: Principles and Practices," J. Systems and Software, vol. 7, no. 4, pp. 259-265, 1987.
[9] Fumiko Nagoya, Shaoying Liu, and Yuting Chen "A Tool and Case Study for Specification-Based Program Review" Department of Computer Science
[10] D.L. Parnas, G. Asmis, and J. Madey, "Assessment of Safety-Critical Software in Nuclear Power Plants," "Nuclear Safety", vol. 32, no. 2, pp. 189-198, Apr.-June 1991.
[11] David Lorge Parnas, "Precise Documentation: The Key To Better Software" middle road software, Inc.
[12] D.L. Parnas, "Tabular Representation of Relations," CRL Report 260, McMaster Univ., Comm. Research Laboratory, TRIO (Telecomm. Research Inst. of Ontario), Oct. 1992.