Volume 3, Issue 9, September 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Improved Clean room Software Engineering using Mutation Testing

Nitika Sharma
Lovely Professional University
Jalandher, India

Abstract -- Cleanroom Software Engineering is a management and technical process that produces high quality software. Cleanroom Software Engineering yields software that is correct by mathematically sound design and that is certified by statistically valid testing. In this paper we propose mutation testing to improve the existing Cleanroom Software Engineering. The main objective of the paper is to make Cleanroom Software Process radical for most developers and focuses on high-value tasks. As improvements on those tasks can yield the most benefit to an organization. To overcome the flaws of Cleanroom Software Engineering new steps have been proposed for improving the technology. Mutation testing helps a user to create test data by interacting with the user to iteratively strengthen the quality of test data. Because program units are so much smaller, testers can find failures more efficiently during unit testing. In addition, it is easier to track down and solve faults in software units.

Keywords:

I. INTRODUCTION

Cleanroom Software Engineering is a set of techniques and practices for the specification, development and certification of software-intensive systems. The Cleanroom method has been used successfully on projects of various sizes and levels of complexity. Management of the Cleanroom process is based on a life cycle of development and certification of a pipeline of user-function increments that accumulate into the final product. Teams in IBM and other organizations that use the process are achieving remarkable quality results with high productivity. The Cleanroom approach as described is an amalgam of some of the most promising software engineering techniques.

Cleanroom is a software engineering development methodology that is claimed can produce near-zero defect software. It comprises a set of practices that cover the software development lifecycle from specification through to testing. Incremental development Systems are produced as a series of successive increments, each of which adds to the functionality of the existing code. Some of the Advantages are:

- Quality.
- Productivity.
- Life cycle costs.
- Return on investment.

To overcome the flaws of Cleanroom Software Engineering new steps have been proposed for improving the technology. Mutation testing helps a user create test data by interacting with the user to iteratively strengthen the quality of test data. Because program units are so much smaller, testers can find failures more efficiently during unit testing.

II. CLEANROOM SOFTWARE ENGINEERING

Cleanroom Software Engineering is a theory based team-oriented engineering process for developing very high quality software under statistical quality control. The Cleanroom process combines formal methods of object-based box structure specification and design, function theoretic correctness verification, and statistical usage testing for reliability certification on to produce software approaching zero defects. Cleanroom software engineering is an engineering and managerial process for the development of high-quality software with certified reliability.

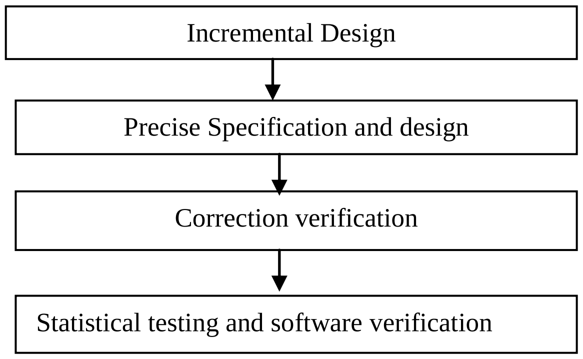
Incremental Design
Precise Specification and design
Correction verification
Statistical testing and software verification

Fig. 1Existing Cleanroom Software engineering

A .Project planning

Cleanroom project planning is robust, with emphasis on an incremental development process for managing requirements, risks, resources, reuse, and other technical factors influencing project success. The Cleanroom Reference Model incorporates additional CMM requirements for effective planning, such as ensuring adequate funding for the planning period itself, using estimates that are derived according to a documented procedure, and gaining explicit agreements from all individuals and groups who have responsibilities related to the project. Requirement analysis software requirements analysis and documentation are essential Cleanroom activities. Requirements management is a major aspect of Cleanroom incremental development. Ongoing confirmation or clarification of requirements occurs through planned customer evaluation of successive increments. New or changed requirements are accommodated through top down evaluation of impacts on all work products at the outset of each increment's development cycle. As with all Cleanroom work products, the requirements document and all modifications to it are subject to peer review and engineering change control.

B. Incremental planning

Cleanroom approach is based on development and certification of a pipeline of user function software increments that accumulate into the final product. Incremental development enables early and continual quality assessment and user feedback, and facilitates process improvements as development progresses. The incremental approach avoids risks inherent in component integration late in the development cycle. Incremental development permits systematic management and incorporation of requirements changes over the development cycle. Incremental development as practiced in the Cleanroom process is based on the principle of referential transparency. Referential transparency means that the only thing that matters about an expression is its value and any sub expression can be replaced by any other that is equal in value. In the context of software development, this property requires that a specification and its design decomposition define the same mathematical function, that is, the same mapping from the domain of inputs to the range of correct outputs [1]. Architectural requirements and risk avoidance strategies place additional constraints on the increment content. For correctness, each increment must satisfy its parent specification through the functions it provides combined with the sub specifications it contains for future increments.

C .Software Reengineering

The purpose of the Software Reengineering Process is to prepare reused software for incorporation into the software product. Non-Cleanroom-developed software can be incorporated into Cleanroom projects. Such software may require reengineering to enable developers to maintain intellectual control and to achieve Cleanroom levels of quality and reliability. Software may be reused as is, reused through interface controllers such as wrappers, or reused after reengineering. Reused software must satisfy two principal Cleanroom requirements. First, the functional semantics and interface syntax of reused software must be understood and documented, to maintain intellectual control and to avoid unforeseen failures in execution.

D. Function Specification and increment Design

A key Cleanroom principle is that programs can be regarded as rules for mathematical functions (or relations). That is, programs carry out transformations from input to output that can be precisely specified as function mappings. Programs can be designed by decomposing their function specifications, and can be verified by abstracting and comparing their designed functions to their function specifications for equivalence. This concept is scale-free, with application ranging from large specifications for entire systems down to individual control structures, and to every intermediate decomposition and verification along the way. Three special types of mathematical functions are important in Cleanroom development because of their correspondence to useful system views, and their interrelationships in a stepwise decomposition and verification process. These functions are represented using Black box, State box and Clear box, collectively called Box Structures and are used by the specification team. Box structures map system stimuli and the stimulus histories into responses.

III. RELATED WORK

A phased implementation of the Cleanroom process enables quality and productivity improvements with an increased control of change. An introductory implementation involves the application of Cleanroom principles without the full formality of the process; full implementation involves the comprehensive use of formal Cleanroom methods; and advanced implementation optimizes the process through additional formal methods, reuse, and continual improvement. The MVS project, the largest IBM Cleanroom effort to date, successfully applied an introductory level of implementation [3]. Cleanroom [10, 2] is a software engineering methodology that challenges the traditional view that zero defect software is either unfeasibly expensive or well-nigh impossible for commercial software systems. It comprises a set of practices that cover the software development lifecycle from specification through to testing. Incremental development Systems are produced as a series of successive increments, each of which adds to the functionality of the existing code.

Shirley Becker and James Whittaker collaborated with other known practitioners to develop a comprehensive set of guidelines for the implementation of Cleanroom. Cleanroom Software Engineering Practices brings together concepts, lessons learned and best practices resulting from Cleanroom projects with which the authors participated in the past several years. This valuable book covers: specification and design of intended behaviour; incremental development process model; stepwise refinement of specifications to code; correctness verification of developed code; statistical certification of compiled software products and much more. Cleanroom Software Engineering is a set of techniques and practices for the specification, development and certification of software-intensive systems. The Cleanroom method has been used successfully on projects of various sizes and levels of complexity but success stories of its use have focused on a particular aspect or on the final results but rarely on its actual implementation.

IV. PROPOSED MUTATION TESTING

Mutation testing, a white box technique introduced to prevent human errors in the software. During mutation testing, faults are introduced into a program by creating many versions of the program ,each of which contains one fault. Test cases are used to execute these faulty programs with the goal of distinuiguishing the faulty programs from the original program .Faulty program are called mutant of the original program and mutanat are killed if the output of the mutant is different from the original problem. Mutant follow coupling effect ,which says that complex faults are coupled into a simple fault in such a way that a test data set that detects all simple faults in a program will detect most complex faults. Mutation analysis provides a test creation, rather than a test process. Attesting creation is a rule or collection of rules that improve requirements on a set of test cases.
Steps for mutation
1. System creates mutant version of the program when the same is submitted .Mutant operator are used to create the mutant ,operator could be, replacing each operand with other legal operand, modify expressions by replacing operand or deleting the statement.

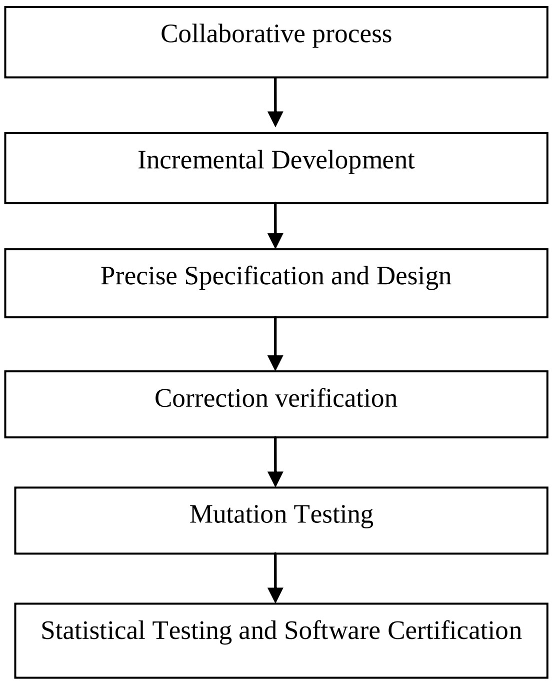

Fig. 2 Proposed Cleanroom Software Engineering.

2. Test cases are supplied to the system to serve as input to the program. Each test case is executed on the original; program and if the output is correct, it is executed on the mutant program.
3. After all the tests cases have been executed the mutation score is computed .The mutation score is the ratio of dead mutant over the number of non equivalent mutants.
4. If mutant are not killed set of test cases are enhanced by supplying new inputs. The mutants that are not killed are called Equivalent mutants. Equivalent mutants are the mutant that produces the same output as the original program in every case so they are not killed.
A mutation score threshold can be set as a policy decision to require testers to test software to a predefined level. Failure in the software are detected when tests cases are executed against the original programs. The tester must decide whether the output of the program on each test case is corrected.

V. CONCLUSION

We have studied various aspects of Cleanroom software engineering. The Cleanroom approach has both technical as well as management control. It is basically used for development of safety critical products and not for ordinary products. Cleanroom practitioners use tables and symbols formalism. Here, usage testing is done to indicate quality. It uses box structures for verification, so no debugging is required. But it cannot adapt quickly to rapidly changing requirements. For improving this aspect of Cleanroom Software Engineering, we suggest the use of mutation testing for requirements engineering. Unit testing is very important for debugging the human errors. But there is no provision for unit testing in Cleanroom Software Engineering. The inclusion of mutation testing can improve this aspect of Cleanroom Software Engineering. The Cleanroom Software Engineering can be further improved by finding the gaps between Case tools defined for the Cleanroom Software Engineering and how they can be used to improve Cleanroom Software Engineering

REFERENCES

[1] Richard C. Linger , Carmen ,Cleanroom Software Engineering ReferencesModel Cersion 1.0 ,Technical Report CMU/SEI -96,November 1996.

[2] Linger ,Mills and Witt ,Structured Programming :Theory and Practice ,Addison Wesley ,1979.

[3] Haulster ,P.A, Linger ,Adopting Cleanroom Software Engineering with a Phased Approach ,IBM Systems Journal ,volume 33,Number 1,1994.

[4] Pfleeger ,S.L ,Hatton ,Investigating the influence of formal methods,IEEE Computer ,Volume 30 ,Issue 2,Feb 1997,PP 33-43.

[5] Roger S. Pressmann ,Software Engineering A Practitioner's Approach fifth edition.

[6] R.H .Cobb and H .D .Mills Engineering Software Under Statistical Quality Control ,IEEE software ,7(6) :1990.

[7] Booch ,Grady ,Object Oriented Analysis and Design ,Addison Wesley ,2nd edition 1994.

[8] Dayer ,Michael ,the Cleanroom Approach to quality software development ,1992.

[9] Cleanroom Software engineering tools http://www.thedecs.com/database/url/key/64/90.htmml

[10] Thomas J.Bergin ,Computer aided software engineering :issues and trands for the 1990s and beyond,Idea Group Inc (IGI) 1993.

[11] A.R .Heyner ,Box structured methods for system development with objects ,ZBM Systems journal 32, 1993.

[12] An Fruhling ,Desgining and evaluating Collabrative Processes afor requirements Elictation and validation Proceedings of the 40th Hawaii International Conference on Sysytem Sciences -2007.

[13] Kemp , formal Methods Spection and verification Guidebook For software and computer Systems,volume I ,NASA 1998.

[14] Chirtian Bucanac ,Object Oriented Testing Report ,1998.

[15] D.P. Kelly ,Im proving Software Quality using Statstical Testing Techniques ,Information and software Technology ,2000.