THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

Cleanroom Software Engineering for Zero-Defect Software

Er. Rasneet Kaur Chauhan
Department of Computer Science and Engineering
Guru Nanak Dev University, RC Gurdaspur

Abstract:
Cleanroom Software Engineering is a process which aims to achieve a zero defect software. As the IT industry is progressing, the need for software which are more compatible and error-free, comes into place. A number of software engineering models have been there to satisfy these needs but the software with almost no defect is highly needed. This is where the process called Cleanroom software engineering comes into picture. It uses statistical quality control method to develop a zero-defect software. This paper aims to give overview of Cleanroom software engineering process , its potential benefits, its weaknesses. We also discuss the practical challenges that it faces.

Keywords*: Cleanroom Software Engineering, Statistical Quality Control, Box Structure specification, Zero Defect*

1. Introduction

A number of processes have been developed to produce new software and to improve their quality. These aim to develop software with almost no defects [1]. But cleanroom software engineering process is one which produces a zero defect software. At first, it may seem impossible. But it really does produce software with zero defect. The Cleanroom process focuses on the design of the program rather than debugging code. Cleanroom software engineering was developed by Harlan Mills and his colleagues, at IBM Corporation's Federal Systems Division during the early 1980s. It emphasizes on defect prevention rather than defect removal. It aims to remove all the defects before the code is being run. Thus it's a highly advanced process that demands high costs and more intensive development. A cleanroom software engineering divides the whole process of developing software into three engineering teams. For formal specification, a team of specification engineers is formed, which break specifications into increments for development and certification. Other team creates software to the specification of these increments with formal specification, but does no debugging and testing and is done by development engineers. The last team of certification engineers test and certify the correctness of those increments by statistical testing. Certifying the correctness of a software requires two conditions , such as:

- Statistical testing with input characteristics of actual usage
- No failures in testing.

The goals of cleanroom software engineering are very high. It focuses to produce a high level, zero defect software. To achieve this goal, highly intense and rigorous process is needs. Thus it aims on bug prevention rather than bug removal. Cleanroom software engineering is a correct, verifiable software using an incremental development process [2]. Cleanroom Software engineering is a practical development and certification approach that uses statistical quality control to reduce software defects and costs. Despite numerous benefits the approach is still not popular because of the cost and extensive amount of work required.

2. Benefits

Some of the benefits that cleanroom software engineering offers are as follows:

- Productivity: Cleanroom software engineering offers better and enhanced productivity than other development models due to rigorous and intensive development process used for development of software and reduced time required to debug and test software.
- Quality: Cleanroom software engineering improves quality of software as it ensures a zero-defect software. It removes each defect from the software and hence the software quality also increases.
- Life Cycle Costs: The cost of life cycle is also less as it needs less maintenance, debugging and testing. Moreover, team reviews are more cost-effective means of improving quality and disseminating knowledge. The adoption cost can be recovered on first project through improved effectiveness in texting and development.
- Maintenance: Maintenance is much easier with cleanroom software engineering. Software developed with this process has understandable, distinct specification with less complicated design, thus resulting in a trouble free maintenance. This happens because this process includes team reviews, box structure method, formal design and certification for creating specifications.

- Reliability: It has high reliability as it has a thorough incremental software process for development of zero imperfection and high reliability software using statistical quality control and certification.
- Errors and Failures: Cleanroom software engineering offers a zero defect software and it is the biggest benefit as it reduces all the errors and failures found during testing. Thus the development cycle time also reduces.
- Less Complicated Design: The box structure generates a less-complicated design. This structure allows a user view black box specification that is in terms of system usage. Thus, these specification are well-defined, clear, understandable and exhaustive and well-defined creating a clear view of software program, resulting in better productive software [11].

3. Disadvantages

The reasons why cleanroom software engineering is not that popular are:

- Too rigorous and intensive approach: Cleanroom software engineering is a very rigorous approach and hence it requires an intensive training. Most of the software industry is working at ad-hoc level of Capability maturity model and thus they don't make proper use of all processes required in all phases of software development life cycle. In order to implement this, an intensive training is required.
- Much theoritical and mathematical: Some research show that cleanroom software engineering is a lot theortical and mathematical and hence causes some difficulty in understanding. Also the correctness proof part can be complicated and time consuming for non-safety-critical programs.
- No Unit Testing: There is no unit testing in this process. But disallowing unit testing is less productive. It doesn't allow unit testing but instead prefers statistical quality control which is causes a wide gap in traditional software development approaches.
- Not widely Accepted: Cleanroom software engineering is much less accepted and less popular because of its mathematical and rigorous approach. Despite of its so many benefits, its not that widely accepted.
- Cost; The most important factor in choosing anything is its cost. Cleanroom software engineering is very pricy. A lot of time and money is spent in the defect prevention of every stage in the development process. The biggest time consumer of the process is the use of statistical methods to ensure quality.

4. Cleanroom Software Engineering Process Model

Cleanroom software engineering process comprises of following activities:

- Software specification:In this, structures specification and precise specification of pipeline of software increments that accumulate into final software product, which includes the statistics of its use as well as its function and performance requirements
- Software development: then, box structured design and functional verification of each increment, delivery for testing and certification without debugging beforehead, and subsequent correction of any failures that may be uncovered during certification.
- Software certification: Lastly, software testing and certification of the software reliability for usage specification, notification to developers of any failures discovered during certification, and subsequent recertification as failures are corrected.

Cleanroom software engineering development process has various phases. The initial phase consists of Project planning, Project management, Performance improvement, and engineering change. Then, all these processes move in increments. Each increment consist of increment planning, requirement gathering, box structure specification, formal design ,correctness verification, code generation, inspection and verification , Statistical use planning , Statistical use testing and certification .This approach uses box structures (black box ,state box and clear box) for design specification. In a Cleanroom development, correctness verification replaces debugging and unit testing[5].

After coding is complete, the software immediately enters system test with no debugging. All test errors are accounted for, from the first execution of the program with no private testing allowed. At the end the function of system testing is to certify the quality of the software with respect to the systems specification. Various phases of CSE are enumerated below:

4.1. Cleanroom Management Processes

The main objective of this phase is to do necessary planning, management and improvements. This is the initial phase of Cleanroom software engineering process which includes the following:

- **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 gaining explicit agreements from all individuals ,ensuring adequate funding for the planning period itself, using estimates that are derived according to a documented procedure, and and groups who have responsibilities related to the project[3]. Requirement analysis software requirements analysis and documentation are essential Cleanroom activities. During this phase, the Cleanroom processes are tailored to meet project specific requirements.
- **Project Management**
  Requirements management is a major aspect of Cleanroom incremental development. The purpose of this activity is to manage the project, to manage resources, abd deliver project on time and that too within budget. This phase is concerned

with the management of Communicating with customer and peer organization, setting up of and training Cleanroom teams, manage planned Cleanroom processes.

- **Performance Improvement**
  In this, continuous evaluation of processes and new technology is introduced. Continuous performance monitoring is also done for continuous improvement. Also, team performance is assessed and enhanced through continuous comparisons and deviations.
- **Engineering amendments**
  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. This phase is concerned with planning and performing additions, changes, and corrections to work products.

4.2. Cleanroom Specification process

The first process of each increment is specification. It consists of following activities:

- **Increment Planning**
  The basic objective of increment planning is forming plans to assign customer requirements specified in the function specification to different software increments that satisfy the software architecture. It is also concerned with defining schedule and distribution of resources for increment development and certification[4]. A incremental plan is made and thus various tasks are managed. The incremental process is mainly concerned with moving from initial to final form through a number of increments that implement user function in the system, thereby leading to the required system.
- **Requirement Gathering**
  Requirement gathering phase is used to define requirements for the software product also includes function, usage, environment, and performance as well as to make an agreement with the customer on the requirements as the basis for functions and usage specifications. In addition to simplifying the customer's notion of his current requirements this phase also assists the customers to identify those requirements which he might not be aware. Thus , if we have requirements notified earlier, then it saves time as well as effort.
- **Box Structure Specification**
  Cleanroom software engineering process uses box structures to describe the behavior of the system. Box structures provide a stepwise refinement and verification methodology for information systems using mathematically based techniques. They structures are specifically helpful for recording and dividing requirements specifications[6]. Black box specifications are verified if they are complete, consistent, and correct. State box specifications are confirmed with respect to black box specifications, and clear box procedures are verified with respect to state box specifications. A number of correctness questions are asked during functional verification. Correctness is done by group agreement and/or by formal proof techniques.
- **Function specification**
  In this, complete functional behaviour of the system is analyzed. In the sequence-based specification process, the purpose is to expand specification into implementation through small steps. The boxes are used for refinement. Formal design uses three system structures for specification and design: black box, state box, and clear box.

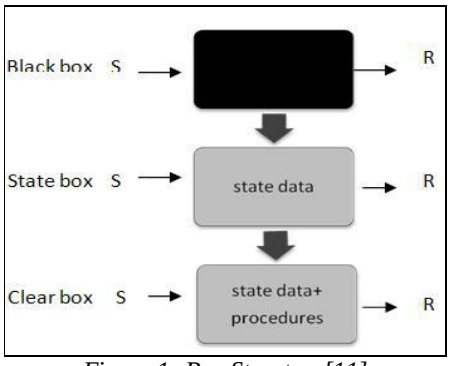

Figure 1: Box Structure[11]

A black box specifies the external behaviour of a system or system component. It specifies the behaviour of system by mapping inputs to outputs. Mostly formal specifications are used [7].

A state box refinement of a black box and specifies how specifications can be turned into implementation. It simplifies a state machine, encapsulates the process implementation, and how the inputs and outputs are represented; data that must be retained between transitions is also encapsulated.

A clear box refinement of a state box and is where process and data implementations are represented. It specifies various procedure designs required to achieve the state box behaviour, and may reuse existing black boxes or introduce new black boxes for subsequent refinement. They have flexibility to be defined in either design language or target language for the system.

A Formal Design begins with an external view (black box), and is transformed into a state machine view (state box), and is fully developed into a procedure (clear box).

4.3. Cleanroom development processes

It is the second process of each increment and it includes the following:

- **Software reengineering**
  Software reengineering process is used to reuse the already existing software by making some changes to it [8]. The software to reengineered must meet some conditions such as, the naming convention should be made more general, the functional semantics and interface syntax if the software to be reused must be properly redocumented. The processes that are reengineered are properly redocumented and entered into inventory for further use.
- **Correctness Verification**
  Correctness verification is done by the development team before release to the certification test team. The correctness of the process is based on substantiating individual control structures like conditions and loops, rather than tracing paths, so those control structures are the ones to be verified. Only a finite number of checks are permitted, and all software logic is to be verified in possible circumstances of use. The verification step is highly successful and innovative in removing defects and developing a better quality software.
- **Code Generation and Verification**
  After the design is complete, the next phase is to develop the code and to verify it. The purpose is to design a software increment and code it such that it conforms to Cleanroom design principles. The box structure specifications which are represented in a specialized language, are coded into an appropriate programming language. Proper care is taken that inspection and verification is done immediately after the code is developed.

4.4. Cleanroom Certification Process

This is the final process of each increment. It occurs after the specification and development processes are completed. Once verification, inspection and usage testing are done the increment is certified as ready for integration. It comprises of following :

- **Using modelling and Statistical Testing**
  Cleanroom software engineering makes use of usage models for statistical testing. Various statistical test cases are made, test environment is primed. This is where software's suitability for use is checked in a format statistical experiment. A set of various test cases that use "probability distribution" of usage are designed and premeditated. Further statistical usage testing is done by executing a series of tests derived from a statistical sample of all probable program executions by all users. A statistical usage model is characterized through a graph, where the nodes indicate events and transitions between events are represented by arcs. Events may be inputs from the user or environment, or abstract states-of-use [9]. Test cases are produced from these models in statistical testing. Then customer reviews all those usage models and agrees that they generate all the scenarios for use. The success or failure of a test case is determined by comparing actual software behavior with the required one. The results of the comparisons make decisions as if to keep on continuing or stopping testing. Usage models are formed incrementally.

Finally, these models are shaped into a final form simultaneously with increment designs. Usage model analysis is helpful in test planning, project tracking and as a management tool.

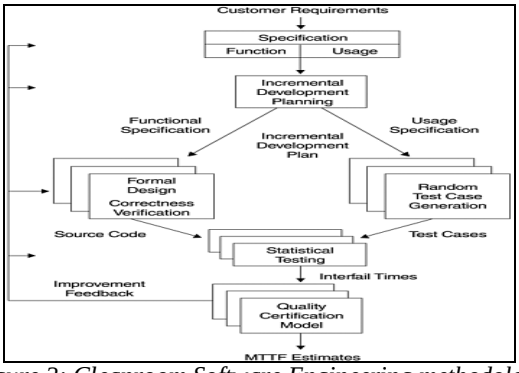

Figure 2: Cleanroom Software Engineering methodology

- **Statistical testing and certification**
  The purpose of statistical testing and certification is to demonstrate a software's performance on a formal dtatistical experiment. Software certification offers a scientific way to verify software appropriateness for usage. Various populations and samples theory are created and hence used for verification. Cleanroom process testing methods are based on the idea that when a system that is too large has a population model that is too large to logically test every case, sampling must be used. The first execution of software is done and hence checked if required results are met and if there are any errors in it, and hence a certification is given to them. The various test cases are made and executed and evaluated. Evaluations and decisions regarding product quality and process control are documented in the Increment Certification Report [10].

Certification Steps:

- Create various usage profiles and scenarios
- Create test cases from these usage profiles
- Execute test cases
- Note down failures[11]
- Record and estimate reliability.

5. Cleanroom Experiences

Cleanroom software engineering practice is not widely practiced but still it has played an important role in following:

- The IBM COBOL structuring facility (IBM COBOL/SF), a complex product of some 80K lines of PL/I source code, was developed in a cleanroom environment.
- A version of U.S. A.F. HH60 (helicopter) flight control program of over 30KLOC was developed using cleanroom.
- The coarse/fine attitude determination subsystems (CFADS) of UARS attitude ground support system (AGSS) of some 30KLOC was developed with cleanroom with NASA.
- Software Engineering Laboratory (SEL) of the NASA Goddard .Space Flight Center.
- Army Weapon Systems[11].
- OS32 Operating System for Telecommunication Switches.
- Industries such as aerospace, telecommunications, graphical, and CASE tools have implemented CSE successfully.

6. Conclusion

Cleanroom Software Engineering has been around for over twenty-five years. It ensures high-quality software with certified-reliability and has led to major improvements over other software methods. The Cleanroom method has evolved throughout the years and has been incorporated in many new software practices.

Cleanroom Software Engineering is a very flexible practice. It can be used in any software project. Because of this, Cleanroom Software Engineering will be around for years to come and will continue to evolve and be incorporated with newer technologies. Cleanroom Software Engineering is essential when you want a high quality product that can improve productivity as well as reduce cost and time.

7. References

1. Foreman, John. (1997). Cleanroom Software Engineering Retrieved March 27, 2006 from http://www.sei.cmu.edu/str/descriptions/cleanroom_body.html
2. Deck, Michael. (1994). Cleanroom Software Engineering: Quality Improvement and Cost Reduction. Retrieved on March 27, 2006 from http://www.cleansoft.com/cleansoft_library.html
3. Linger, Richard C., Trammell, Carmen J. (November 1996). Cleanroom Software Engineering: Reference Model, Version 1.0. Retrieved March 27, 2006.
4. http://www.sei.cmu.edu/pub/documents/96.reports/pdf/tr022.96.pdf
5. Pressmen, Roger S (2004),"Software Engineering: A Practitioner's Approach", 6th ed., McGraw-Hill, New York
6. Ananthpadmanabhan Harish Chetan, Kale Mujtaba, Khambatti Ying, Jin Taufiq, Usman Shaun Shu, Zhang," Cleanroom Software Development", [Online] Available: http://khambatti.com/mujtaba/ ArticlesAnd Papers/Cleanroom% 20Software%20Development.pdf
7. Cleanroom Software Engineering. Retrieved from University of Texas at Arlington website:
8. http://www.uta.edu/cse/levine/fall99/cse5324/cr/clean/page1.html
9. DACS, The Data and Analysis Center for Software. Retrieved January 5, 2009 Found at: www.dacs.dtic.mil/databases/url/key.php?keycode=64
10. Prowell, Stacy et al. (1993). "Cleanroom Software Engineering: Technology and Process".Reading, MA: Addison Wesley Longman, Inc.
11. Kamaldeep Kaur(2011). "Cleanroom Software engineering: towards high reliability software"