VIKRAM SRI NITESH TANTRAVAHI
13247361

Cleanroom Software Engineering Process

College of Computing & Informatics
Drexel University

Research Paper

1. INTRODUCTION

Cleanroom software engineering process is a procedure that develops software with a certifiable level of reliability. Harlan Mills developed the Cleanroom process in the early 1980s at IBM along with his colleagues. It is a theory-based, team oriented software development process and a certification of high-reliability software systems under statistical quality control. The main objective of the Cleanroom process is developing software that exhibits zero failures while in use. It is developed using the formal methods, correctness verification and Statistical Quality Assurance (SQA). The way of approach behind this process is based on the mathematical methods of software specification, design and correctness verification with statistical methods and testing based on the usage to certify the software fitness. Cleanroom is a modern approach to software development. The Cleanroom process has evolved pretty well over last few years can prove that it can enhance and improvise both the productivity of developers who use it and the software quality they produce.

2. GOALS OF THIS EFFORT

The main goal of the Cleanroom process is to develop software, which has no defects or errors or bugs right after the development itself i.e., even before any testing is performed. The main ideology is to get it right the first time itself. The actual process of this is that the software product quality is to be improved by verifying the correctness. There are few targets or goals that this process is aiming to achieve that are mentioned below.

2.1 Manageable Development

Maintenance of intellectual control of software development by the managers and the technical teams is a very important aspect in software development. This is possible by the Cleanroom methods that are implemented. It is necessary that at every stage in the development cycle, the progress of the work should be stated very clear and distinct. The work products are to be accumulated into the final product in a predictable fashion throughout the development and the product integrity is maintained throughout its life. The teamwork that is based on well-defined engineering process is required by the intellectual control. It results in managing complexity, reducing risks, avoiding rework and meeting the business objectives for the planned schedule and respective budget performance.

Intellectual control depends on the technologies employed by the development teams for a better outcome of results. Sometimes, nothing works and no one can be blamed and not even a good management seems to set things right. These frustrating and perplexing muddles in software development are usually a result of inadequate technologies and processes. Methods for precise specification and design, correctness verification, usage testing, and measurement of software quality and reliability are provided by Cleanroom. There are few theoretical foundations such as mathematical completeness and consistency, verifiable correctness, and traceability among work products that are essential for project manageability and success of products in which the Cleanroom is rooted in depth.

2.2 No Failures in Use

The software failures are usually treated in today's world as inevitable and uncontrollable and many defect correction techniques are implemented to reduce there in the software development process. Most of the software organizations follow the postproduction defect correction that involves many substantial operations that drain the productivity and profitability of the system. The tangible costs of software failures tracking problems, finding and correcting defects, distributing fixes, and soon are not quantified in most organizations, and are far greater than most people imagine. The intangible costs of software failures in diminished customer confidence and loyalty are difficult to quantify, but obviously drive the total cost even higher.

Most of the software failures cannot be avoided. Software failures are the result of ineffective specification and development practices that permit introduction and survival of defects, and testing practices that permit defects to remain undetected, only to be discovered in field use. In Cleanroom, development teams use rigorous specification, design, and verification practices, coupled with testing practices that provide valid measures of development performance in approaching the goal of zero defects. The payoff is improved manageability, reduced rework, and sharp reductions in direct and opportunity costs of defect correction over the market lifetime of products.

3. DEFINING ELEMENTS OF CLEANROOM

There is a set of defining elements or major technologies that uniquely characterize the Cleanroom Software Engineering. There are no specific constraints or conditions that are to be followed to use these technologies. They can be used in any comfortable way one needs and in any order that leads to betterment of software practice.

3.1 Incremental Development

A pipeline of software increments that accumulate into the final system are developed and certified during the process of Cleanroom process. The increment development and certification of such operational user functions is done by small, independent teams or teams of teams for large products depending on the size of the software system and the boundary of the project.

These increments are integrated in a top down approach. The functionality grows with addition of successive increments as a continuous process. In this approach, the harmonious operation of future increment at the next level of refinement is predefined by increments already in execution, thereby minimizing interface and de- sign errors and helping developers maintain intellectual control.

Developing the right product with a high quality as a quick rate is the main idea. Early assessment of product quality is given by the incremental development and gives continuous feedback to management as to the progress of development. Team members thus focus on only a portion of the work at any given time rather then trying to keep all things in mind at once.

Incremental development plan organizes a Cleanroom project into an orderly sequence of development cycles, with some amount of end user function developed in each cycle. The Cleanroom team first analyzes and clarifies customer requirements, with substantial user interface and feedback. At the end of each increment, the evolving product can be demonstrated to the customer. The requirements can be reconfirmed and clarified in an informed way minimizing the surprises both ways with such stimulating visibility into the project.

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.

3.2 Functional Specification

The purpose of functional specification is to specify the complete functional behavior of the software in all possible circumstances of use and to obtain agreement with the customer on the specified function as the basis for software development and certification. The specification team creates the Functional Specification document to satisfy the software requirements.

The design characteristics of the functional specification has an external component known as the *black box*, that is transformed into a state machine view called the *state box*, and is fully developed into a procedure called the *clear box*. These are all known as *box structures* altogether which are equivalent in terms of operation. The box structures are object based and support key software engineering principles of information hiding and separation of concerns. Box structures provide an economic incentive for precision. Initial box-structure specifications often reveal gaps and misunderstanding in customer requirements that would ordinarily be discovered later in development at high cost and risk to the project. They also address the two engineering problems associated with system specification: defining the right function for users and defining the right structure for the specification itself.

There are three principles that define the use of box structures. They are:

- All data defined in a design is encapsulated in boxed.
- All processing is defined by using boxes sequentially or concurrently.
- Each box occupies a distant place in a system's usage hierarchy.

3.2.1 Black Box

The black box gives an external description of the external behavior of a system or system part in terms of a mathematical function from stimulus histories to responses. The black box is the most abstract description of system behavior and can be considered as a requirements specification for a system or system part. Its user may be a person or another object.

A black box accepts a stimulus (S) from a user and produces a response (R). Each response of a black box is determined by its current stimulus history (SH), with a black-box transition function:

$$(S, SH) \rightarrow R$$

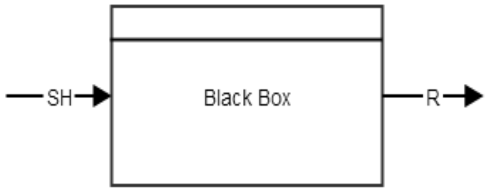

Fig1: View of a Black Box

The main objective of the black-box specification is to define the responses produced for every possible stimulus and stimulus history, including erroneous and unexpected stimuli. By defining behavior solely in terms of stimulus histories, black-box specifications neither depend on nor prematurely define design internals. It is a state-free, procedure-free representation of a function, and the mapping must be complete, consistent, and traceably correct.

3.2.2 State Box

An object's state box is derived from its black box by identifying the elements of stimulus history that must be retained as state data between transitions to achieve the required black-box behavior. A state box defines the required behavior of a system or system part as a mapping (transition) from the current stimulus and old state to the corresponding response and new state. After state data have been defined, it is no longer necessary to consider the stimulus history.

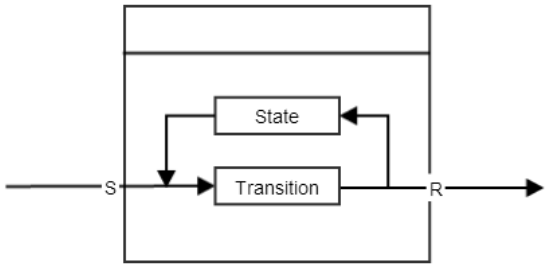

Fig2: View of State Box

The transition function of a state box is

(S, OS) → (R, NS)

where OS and NS represent old state and new state.

As in the traditional view of objects, state boxes encapsulate state data and services (methods) on that data. In this view, stimuli and responses are inputs and outputs, respectively, of specific state-box service invocations that operate on state data.

3.2.3 Clear Box

The clear box replaces the internal black box of the state box with the designed sequential or concurrent usage of other black boxes as subsystems. These new black boxes are in turn expanded at the next level of the system box structure usage hierarchy into state box and clear box forms.

It is derived from its state box by defining a procedure to carry out the state-box transition function. The transition function of a clear box is:

(S, OS) → (R, NS) by procedure

So a clear box is simply a program that implements the corresponding state box. A clear **box** may invoke black boxes at the next level, so the refinement process is recursive, with each clear box possibly introducing opportunities for defining new objects or extensions to existing ones.

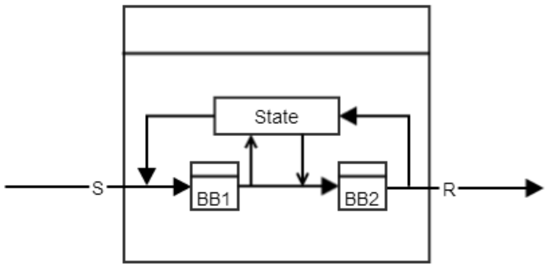

Fig3: Conceptual view of Clear Box

The figure depicts the clear box as a sequential control structure. Alternation (branching), iteration (looping), and concurrent control structures can appear in clear boxes as well. Any new boxes introduced are subsequently refined into state and clear boxes.

3.3 Design and Verification

In box structure specification and design, a black box is defined to record required behavior, then a state box is refined from the black box to define required state data, and finally a clear box is refined from the state box to define required processing. Each box structure is subject to correctness verification in development team reviews.

The team verifies the correctness of each refinement step with respect to the previous step using reasoning based on function theory. In other words, the development team confirms that the stimulus- response mapping defined in one step is preserved in each subsequent step.

Design Refinement: In designing clear-box procedures, you define an intended function, and then refine it into a control structure and new intended functions. In essence, clear boxes are composed of a finite number of control structures, each of which can be checked for correctness.

Design simplification is an important objective in the stepwise refinement of clear boxes. The goal is to generate compact, straightforward, verifiable designs.

Correctness Verification: All Cleanroom-developed software is subject to function-theoretic correctness verification by the development team prior to release to the certification test team. A practical and powerful process, verification permits development teams to verify completely the correctness of software with respect to specifications. The verification step is remarkably effective in eliminating defects, and do Cleanroom teams achieve a major factor in the quality improvements.

3.4 Quality Certification

When we have too many items to test, it become an exhaustive process and then Statistical quality control is used. If that's not the case, you have to statistically sample and analyze and understand some items to obtain a scientific assessment of the quality of all items. This technique is widely used in the manufacturing, in which items on a production line are sampled, their quality is measured against a presumable perfect design, and the sample quality is extrapolated to the entire production line, and flaws in production are corrected if the quality is too low.

Models of safety-critical usage, hazardous usage, malicious usage, or other special usage circumstances are developed when required for focused testing on high-consequence functions. Statistical usage testing can be readily combined with other forms of testing.

The Cleanroom quality certification proceeds in parallel with development in three steps:

a. *Specify usage-probability distributions.*

   Usage-probability distributions define all possible usage patterns and scenarios, including erroneous and unexpected usage, together with their probabilities of occurrence. They are defined on the basis of the functional specification and other sources of information, including interviews with prospective users and the pattern of use in prior versions.

b. *Derive test cases that are randomly generated from usage-probability distributions.*

   Test cases are derived from the distributions, such that every test represents actual use and will effectively rehearse user experience with the product. Because test cases are completely prescribed by the distributions, producing them is a mechanical, automatable process.

c. *Execute test cases, assess results and compute quality measures.*

   At this point, the development team has released verified code to the certification team for first-ever execution. The certification team executes each test case and checks the results against system specifications. The team records execution time up to the point of any failure in appropriate units.

4. CLEANROOM SOFTWARE ENGINEERING REFERENCE MODEL

The Cleanroom Software Engineering Reference Model (CRM), is usually represented in terms of set of 14 Cleanroom processes, which results into 20 work products. Some of the most important features like project management and performance, process assessment and improvement, and technology transfer and adoption are guided by this CRM. Its scope is software management, specification, development, and testing (certification). It is a high level template that must be tailored for use by a specific organization or project. In the Cleanroom Reference Model, each and every process and the work product are supposed to be elaborated through the implementation procedures that address specific organizational or project environments and specific unique development requirements. The above-mentioned 14 processes constitute and form as the base of the Cleanroom technologies, and are explained below with respective descriptions.

4.1 Cleanroom Management Process

This process is mainly present to tailor the Cleanroom Software Engineering processes. There are few more processes under the management process, which are briefly explained further.

4.1.1 Project planning

It helps us define and document plans for a Cleanroom project. It engineers the project, review the plans with the customer, the project team, and the peer groups to extract and produce the software project.

4.1.2 Project Management

It helps and guides the team of the software product on how to manage the Cleanroom Incremental Development and the certification process. This management is very important in order to deliver the software and its associated work products as per the schedule and within the budget. The defining process of the quality objectives and the team performance expectations can be established and the Cleanroom teams can be trained with this process. It also plays a major role in initiating and keeping track of the Cleanroom processes with a software project.

It is very crucial to meet the process performance standards and product quality objectives, and also to improve and enhance the team performance. The management decision-making forms its objective from the statistical testing and certification of successive increments that are produced by the use of quantitative measurements of product and process performance produced are also operated by this project management process of a Cleanroom.

4.1.3 Performance Improvement

This helps in in-depth understanding of the Software Development Plan, processing control standards, and the casual analysis of the failure of software, helps to formally evaluate and improve the Cleanroom team performance continuously. There are always continuous amendments and improvements that are made in the software processes and the tools, which are analyzed and at a later stage introduced to the project for appropriate results.

4.1.4 Engineering Change

This process works according to a protocol that stores the integrity and correctness of the system to correct and change the software system and other work products that are associated with it. It also helps in engineering a change control for all the changes that take place within a system over a period of time. The highest level of specification or design affected by a change is identified as the starting point for any re-specification, redesign, re-verification, or re-certification, as well as any other revision activity.

4.2 Cleanroom Specification Process

4.2.1 Requirement Analysis

To define and analyze the customer requirements at the initial stages for a software system and also the changing requirements that arise with the evolving increments. Requirements analysis may identify opportunities to simplify the customer's initial product concept and to reveal requirements that the customer has not addressed. It enables us to express the requirements in terms of the user and also the customer can review it for the agreement as the basis for the function and the usage specification.

4.2.2 Function Specification

It defines and make sure that the required external behavior of a software system in all possible circumstances of use is based on the Software Requirements. It checks whether the function specification is complete, consistent, correct, and traceable to the software requirements. From the customer point of view, the customer agrees to the function specification on the basis of software development and certification. This expresses the specification in the box structure form and the requirements in a mathematically precise, complete and consistent form.

4.2.3 Usage Specification

It defines all the classes of the users, major patterns of the usage, and the usage environments to establish and analyze the highest-level structure and probability distribution for software usage models, and to obtain an agreement with the customer on the specified usage as a basis for the software certification.

4.2.4 Architecture Specification

There are three key dimensions of the architecture. They are the conceptual architecture, module architecture and the execution architecture. The main purpose is to define these three key dimensions. It analyzes the architectural assets and declares the architectural strategy for the software product development. The history-based black box Function specification that is decomposed into a state based state box and procedure based clear box specifications explains the Cleanroom aspect of the architecture specification. It is the early stage of a referentially transparent decomposition of the function specification box into a box structure hierarchy, which will also be used at the time of increment development.

4.2.5 Increment Planning

This creates and incremental development and certification plan for the software product such that the increments implement user function, accumulate into the final system, execute in the system environment, and permit the systematic feedback on process control and product function and quality. A software system grows from initial to final form through a series of increments that implement user function, executes in the environment of the system, and accumulates into the final system in the incremental process.

4.3 Cleanroom Development Process

4.3.1 Software Reengineering

The purpose of this is to prepare the reused software for incorporation into the software product. The functional semantics and interface syntax of the reused software are to be well understood and properly documented to maintain intellectual control and to avoid and unexpected failures in the execution of the software system. If they are incomplete or insufficient, they can be recovered through correctness verification and function abstraction. The use of usage models and the statistical testing will help us to determine the fitness of the reused software.

4.3.2 Increment Design

Design and code the increments for a software product through stepwise decomposition of box structures, typically from stimulus history-based black box specifications into state-based state box specifications, and then into procedure-based clear box designs containing lower level black boxes for further refinement. Prepare designs for correctness verification by embedding intended function definitions that specify the effect on data of corresponding control structure decompositions. The design is performed in such a way that it is provably correct using mathematical models. Treating a program as a mathematical function can do this. Note that specification and design are developed in parallel, resulting in a box structure hierarchy affording complete traceability.

4.3.3 Correctness Verification

At this stage, mathematically based techniques are used to verify the correctness of a software increment. This is usually done by carrying out correctness verification designs, mostly through verbal proofs of correctness in team views. For instance, the black box specifications are verified to be complete, consistent and correct. This is also carried out to identify and correct the software defects prior to the first state of execution. The State box specifications are verified with respect to black box specifications, and clear box procedures are verified with respect to state box specifications. So the functional verification includes a set of questions for correctness. It is also important to re-verify the work after every any kind of change that happens to take place.

4.4 Cleanroom Certification Process

4.4.1 Usage Modeling and Test Planning

Now at this stage, the usage models are created for the purpose of software testing, defining test plans, and certification. These models that are generated are expressed in terms of probabilities of transition between the individual states and the software usage states. These models can also be created to satisfy few objectives of the software system. For example, expected operational usage certification or certification for use of rarely used functions that are having high probability of failure. The usage model statistics are employed to provide a deeper understanding into the system complexity and also the quality objectives testing that has to be met with adequate amount of testing efforts. The test environment is prepared, a statistical test plan is developed and the statistical test cases are generated.

4.4.2 Statistical Testing and Certification

The software fitness is usually defined in terms of meeting the goals with respect to usage models and certification goals. This software fitness available for use is demonstrated in a formal statistical experiment. In this process, software increments go through the first execution. The statistical test cases are executes under experimental control and the results are evaluated. Here, if there are any failures that are encountered, then the engineering change of activity is initiated. These test cases that are developed here are considered as a success or a failure depending upon the comparison of values of certification measures that are derived from statistical testing with certification goals to assess the software's fitness for use i.e., comparing the actual existing software behavior with the required behavior. The further stages like continuing the testing process, stopping the testing to engineer changes, stopping testing for re-engineering and re-verification or final software certification are all dependent directly on the comparison results.

5. CONCLUSION

Cleanroom software engineering allows errors to be found earlier in the lifecycle, which minimizes expensive rework later on and speeds time to market. Designs are simplified, straightforward, and verifiable, resulting in less "spaghetti" code. Quality is achieved by design and verification, not testing. This built in quality lowers the overall cost of the product, and the designs also tend to be more concise and compact than average; always a good thing for embedded developers. Cleanroom supports prototyping, object orientation and reuse. The technology is platform and language independent. And productivity is high. The methodology is based on structured programming and is compatible with reuse. It is a formal, disciplined process, and composed of a set of stepwise refinements or transformations from the requirements to the code, where each transformation is verified to the previous level of refinement in order to minimize errors.

Cleanroom can be applied to new systems as well as existing systems. For example, poor quality sections of software in existing systems can be re-engineered using certain Cleanroom techniques such as formal correctness verification.

As per my overall assessment on this topic, the pros and cons are as described below.

Advantages:

1. Some of the benefits through the use of Cleanroom are like, the code verification becomes much faster and a deeper understanding of the requirements, design and code.

2. The more the time an individual or a company spends on to come up with a sound design, the less time one will spend translating such design into code.

3. The programs that are built with it are of much more higher quality than compared to the programs built without the process.

4. There is a very high probability of low rate of

defects in a program attributed to the way that Cleanroom uses statistical usage testing.

5. This methodology is incremental so it can be implemented gradually.

Disadvantages:

1. The time it takes to train a team to the point where they are effective and efficient at verifying designs might be too long and expensive.

2. The Cleanroom is something very different from the usual stuff and very out of the norm, and so extreme, that the required drastic change of culture would be so much for the companies to handle.

3. The resources required to train the staff to this level of standard would be beyond the acceptable level.

4. High risks and challenges involved for a company to just try out this process.

6. REFERENCES

[1] R. Linger, H. Mills**,** "A Case Study in Cleanroom Software Engineering: The IBA4 COBOL Structuring Facility," *Proceedings of the 12th International Computer Science and Applications Conference,* IEEE Computer Society Press, October, 1988.

[2] *A Success Story at Prait* & *Whitney: On Track for the Future with IBM's VS COBOL I1 and COBOL Structuring Facility,* publication GK20-2326, IBM Corporation, White Plains, NY, 1989.

[3] A. Kouchakdjian. "Evaluation of the Cleanroom Methodology: A Case Study *at* NASA/Goddard," *Proceedings* of *the Seventh International Conference on Testing Computer Software,* June 18-20, 1990, San Francisco, CA.

[4] Oshana, Robert; An Industrial "Application of Cleanroom Software Engineering - Benefits through Tailoring", 1998.

[5] Kelly, D. P., & Oshana, R. S. (1996, November). "Integrating Cleanroom Software Methods Into an SEI Level 4-5 Program." *U.S. Air Force's Software Technology Support Center.*

[6] Stavely, A. M. (1999, March 22-24). "High-Quality Software Through Semiformal Specification and Verification." Paper presented at the 12th Conference on Software Engineering Education and Training, New Orleans, Louisiana.

[7] Head, G. E. (1994, June 1). "Six-sigma Software Using Cleanroom Software Engineering Techniques." *Electric Library: Hewlett-Packard Journal.*

[8] Deck, M. (1997, May 27-30). "Cleanroom Software Engineering Myths and Realities." Paper presented at the Quality Week, San Francisco, CA.

[9] Deck, M. (1994, October). "Cleanroom Software Engineering: Quality Improvement and Cost Reduction". Paper presented at the Proc. 12th Pacific Northwest Software Quality Conference.

[10] Henderson, J. (1995, May). "Why Isn't Cleanroom the Universal Software Development Methodology?" *U.S. Air Force's Software Technology Support Center.*

[11] Richard C. Linger, "Cleanroom Software Engineering for Zero-Defect Software", IBM Cleanroom Software Technology Center.

[12] Foreman, J. (1997, October 27). "Cleanroom Software Engineering: Software Technology Review." *Carnegie Mellon Software Engineering Institute*