MCA PART II

Paper-XI

Topic: CLEANROOM SOFTWARE ENGINEERING

Prepared by: Dr. Kiran Pandey

(School of Computer Science)

Email-id: kiranpandey.nou@gmail.com

Introduction

Cleanroom software engineering involves the integrated use of software engineering modeling, program verification, and statistical software quality assurance. It is an engineering approach which is used to build correctness in developed software. The main concept behind the cleanroom software engineering is to remove the dependency on the costly processes. The cleanroom software engineering includes the quality approach of writing the code from the beginning of the system and finally gathers into a complete a system.

Under cleanroom software engineering, the analysis and design models are created using a box structure representation (black-box, state box, and clear box). A box encapsulates some system component at a specific level of abstraction. Correctness verification using formal methods is applied once the box structure design is complete. Use of formal methods reduces the number of specification errors dramatically, which means that the customer will encounter fewer errors when the product is deployed. Once correctness has been verified for each box structure, statistical usage testing commences. This involves defining a set of usage scenarios and determining the probability of use for each scenario. Random data is generated which conform to the usage probabilities. The resulting error records are analyzed, and the reliability of the software is determined for the software component.

Cleanroom Strategy

The Cleanroom approach to software development is based on five key strategies:

- *Formal specification:* The software to be developed is formally specified. A state-transition model which shows system responses to stimuli is used to express the specification.
- *Incremental development:* The software is partitioned into increments which are developed and validated separately using the Cleanroom process. These increments are specified, with customer input, at an early stage in the process.
- *Structured programming*: Only a limited number of control and data abstraction constructs are used. The program development process is a process of stepwise refinement of the specification. A limited number of constructs are used and the aim is to apply correctness-preserving transformations to the specification to create the program code.
- *Static verification*: The developed software is statically verified using rigorous software inspections. There is no unit or module testing process for code components.
- *Statistical testing of the system*: The integrated software increment is tested statistically, to determine its reliability. These statistical tests are based on an operational profile which is developed in parallel with the system specification.

The figure below shows the cleanroom process:

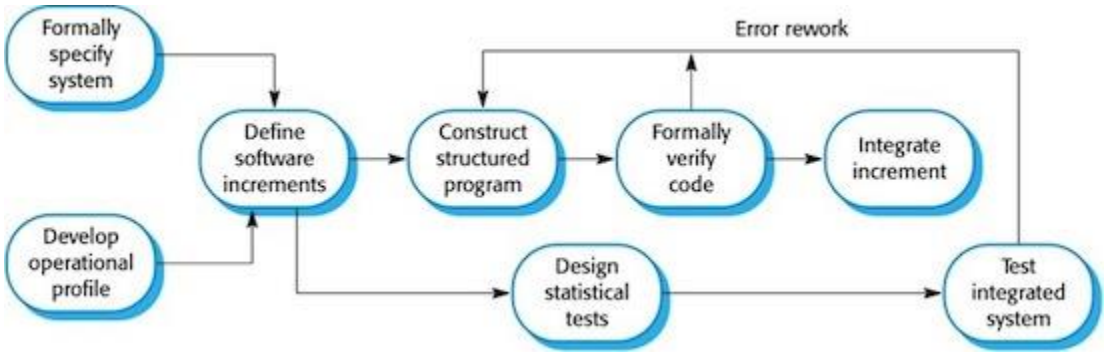

Figure 1: The Cleanroom process

Tasks in cleanroom engineering

1. Incremental planning

- In this task, the incremental plan is developed.

- The functionality of each increment, projected size of the increment and the cleanroom development schedule is created.
- The care is to be taken that each increment is certified and integrated in proper time according to the plan.

2. Requirements gathering

- Requirement gathering is done using the traditional techniques like analysis, design, code, test and debug.
- A more detailed description of the customer level requirement is developed.

3. Box structure specification

- The specification method uses box structure.
- Box structure is used to describe the functional specification.
- The box structure separate and isolate the behaviour, data and procedure in each increment.

4. Formal design

- The cleanroom design is a natural specification by using the black box structure approach.
- The specification is called as state boxes and the component level diagram called as the clear boxes.

5. Correctness verification

- The cleanroom conducts the exact correctness verification activities on the design and then the code.
- Verification starts with the highest level testing box structure and then moves toward the design detail and code.
- The first level of correctness takes place by applying a set of 'correcting questions'.
- More mathematical or formal methods are used for verification if correctness does not signify that the specification is correct.

6. Code generation, inspection and verification

- The box structure specification is represented in a specialized language and these are translated into the appropriate programming language.
- Use the technical reviews for the syntactic correctness of the code.

7. Statically test planning

- Analyzed, planned and designed the projected usages of the software.
- The cleanroom activity is organized in parallel with specification, verification and code generation.

8. Statistical use testing

- The exhaustive testing of computer software is impossible. It is compulsory to design limited number of test cases.
- Statistical use technique execute a set of tests derived from a statistical sample in all possible program executions.
- These samples are collected from the users from a targeted population.

9. Certification

- After the verification, inspection and correctness of all errors, the increments are certified and ready for integration.

Certification Steps

1. Usage scenarios must be created.
2. Usage profile is specified.
3. Test cases generated from the usage profile.
4. Tests are executed and failure data are recorded and analyzed.
5. Reliability is computed and recorded.

Cleanroom Certification Models

- **Sampling model** - determines the number if random cases that need to be executed to achieve a particular reliability level.
- **Component model** - allows analyst to determine the probability that a given component in a multi-component system fails prior to completion.
- **Certification model** - projected overall reliability of system.

Cleanroom process model

- The modeling approach in cleanroom software engineering uses a method called box structure specification.
- A 'box' contains the system or the aspect of the system in detail.

- The information in each box specification is sufficient to define its refinement without depending on the implementation of other boxes.

The cleanroom process model uses three types of boxes as follows:

1. Black box

- The black box identifies the behavior of a system.
- The system responds to specific events by applying the set of transition rules.

2. State box

- The box consist of state data or operations that are similar to the objects.
- The state box represents the history of the black box i.e the data contained in the state box must be maintained in all transitions.

3. Clear box

- The transition function used by the state box is defined in the clear box.
- It simply states that a clear box includes the procedural design for the state box.

Cleanroom Design Concepts

Design Refinement

- Each clear box represents the design for a procedure required to accomplish a state box transition implemented using structured programming constructs
- At each level of refinement the cleanroom team performs formal correctness verification which involves attaching a set of correctness conditions to each structured programming construct
  - Sequence (f → g, h) is refined into:
    Does g followed by h do f?
  - Conditional (p → if <c> then q else r) is refined into:
    Whenever <c> is true does q do p and

    Whenever <c> is false does r do p
  - Loop (m → while <c> do n) is refined into:
    Is termination guaranteed?

    Whenever <c> is true does m followed by n do m and

    Whenever <c> is false does skipping loop still do m

Design Verification

- Reduces verification to a finite process
- Improves quality
- Allows cleanroom teams to verify every line of code
- Results in near zero levels of defects
- Scales up to larger systems and higher levels
- Produces better code than unit testing

Formal Methods

- Define the data invariant, state, and operations for each system function
  - *Data invariant* -condition true throughout execution of function that contains a collection of data
  - *State*- stored data accessed and altered by function
  - *Operations* - system actions that take place when data are read or written to the state (an invariant, a precondition and a post condition are associated with each operation)
- Specification is represented in some set theoretic type notation from some formal language (e.g. Z or OCL)
- Specification correctness can be verified using mathematical proofs (set operations, logic operations, sequences, induction)

Writing Formal Specifications

- Begin by defining state in terms of abstract items to be manipulated by the function (similar to variable declaration in a programming language)
- Define the data invariant by writing the data relations that will not change during the execution of the function using mathematical notation
- Write the precondition and post condition for the function using mathematical notation to show the system state before and after function execution

Formal Specification Language Components

- Syntax that defines the specific notation used to represent a specification
- Semantics that help to define the universe of objects that will be used to describe the system
- Set of relations that define the rules that indicate which objects properly satisfy the specification

Object Constraint Language (OCL)

- Notation developed to allow users to add more precision to UML specifications

- OCL is a modeling language that has all the attributes of a formal language
- Typically one begins with a set of UML diagrams (class, state, activity)
- OCL Boolean expressions (constraints) are created for the diagram elements
- OCL constraint expressions involve operators operating on objects
- Any implementation derived from UML model must ensure that each constraint always remains true
- OCL can be used to specify the preconditions and post conditions for operations

Z Specification Language

- Applies typed sets, relations, and functions within the context of first-order predicate logic to build schemas
- Schemas are box-like structures that introduce variables and specify the relationships between these variables
- Schemas are the formal specification analog of a programming language component
- Schemas describe the stored data used to define the state of a system and describes what data the operations alter to define a new state