CHAPTER 29 CLEANROOM SOFTWARE ENGINEERING

Overview

This chapter discusses the cleanroom approach to software engineering. The philosophy behind cleanroom software engineering is to develop code increments that are right the first time and verify their correctness before testing, rather than relying on costly defect removal processes. Cleanroom software engineering involves the integrated use of software engineering modeling, program verification, and statistical software quality assurance. 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 is applied once the box structure design is complete. 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.

Distinguishing Characteristics of Cleanroom Techniques

1. Makes explicit use of statistical quality control
2. Verifies design specification using mathematically-based proof of correctness
3. Relies heavily on statistical use testing to uncover high impact errors

Reasons Cleanroom Techniques Not Used Widely

1. Some people believe cleanroom techniques are too theoretical, too mathematical, and too radical for use in real software development
2. It does not advocate unit testing, relying instead on correctness verification and statistical quality control (a major departure from the way most software is developed today)
3. Since most of the software industry is operating at the lower levels of the Capability Maturity Model, most organizations do not make rigorous use of the defined processes needed in all phases of the software life cycle

It should be noted that all of the above roadblocks to cleanroom usage can be overcome and that cleanroom software engineering offers substantial benefits to those who do it.

Cleanroom Strategy

- **Increment planning**. The project plan is built around the incremental strategy.
- **Requirements gathering**. Using Chapter 11techniques, customer requirements are refined for each increment.
- **Box structure specification**. Box structures isolate and separate the definition of behavior, data, and procedures at each level of refinement.

- **Formal design**. Specifications (black-boxes) are iteratively refined to become architectural designs (state-boxes) and component-level designs (clear boxes).
- **Correctness verification**. Correctness questions are asked and answered and followed by formal mathematical verification when required.
- **Code generation, inspection, verification**. Box structures are translated into program language; inspections are used to ensure conformance of code and boxes, as well as syntactic correctness of code; followed by correctness verification of the code.
- **Statistical test planning**. A suite of test cases is created to match the probability distribution of the projected product usage pattern.
- **Statistical use testing**. A statistical sample of all possible test cases is used rather than exhaustive testing.
- **Certification**. Once verification, inspection, and usage testing are complete and all defects removed, the increment is certified as ready for integration.

Box Types

- **Black box** - specifies a set of transition rules that describe the behavior of system components as responses to specific stimuli, makes use of inheritance in a manner similar to classes
- **State box** - generalization of a state machine, encapsulates the data and operations similar to an object, the inputs (stimuli) and outputs (responses) are represented, data that must be retained between transitions is encapsulated
- **Clear box** - contains the procedural design of the state box, in a manner similar to structured programming

Design Verification Advantages

- 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

Statistical Use Testing

- Attempts to test software in the way that it is intended to be used by users
- Tester determines a usage probability distribution for the software
- Test cases are generated for each set of use case stimuli based on the usage probability distribution
- The sequence of test cases is also determined by the usage probability distribution, these test case sequences will match the use cases
- The test team executes these test case sequences and verifies software behavior against the system specification

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