Cleanroom Process Model

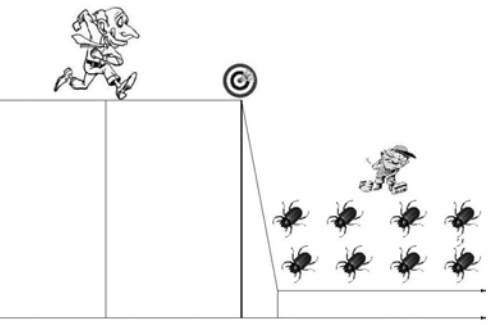
Presented by
Chaman Singh Verma

24th October 2003

Outline

- Introduction
- Philosophy
- Process
- Success Stories
- Discussions

Traditional Software Development At Work.



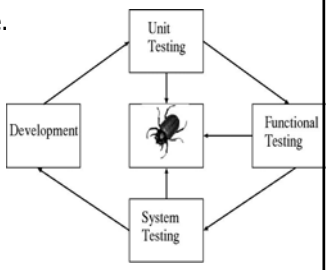
Cleanroom:

Cleanroom

- Manufacturing areas in microelectronics, pharmaceutical and biotechnology allow only zero-tolerant contamination in the room.
- Everything is suspected even your own body, air-conditions, fans, your shoes, even your formal clothes.
- Everyone has to pass through strictly controlled chambers to make sure that everything is de-contaminated.

Bug-Centered Computing

- Defects are inevitable.
- Defect removal are part of software development.



Are bugs in software unavoidable ?

- Bugs are symptoms of mismanagement and unpredictable environment.
- It is difficult to eliminate them, but good software engineering can control their insatiable desire for reproduction.

Let us understand how ***Cleanroom Process*** accomplishes that.

Cleanroom Philosophy:

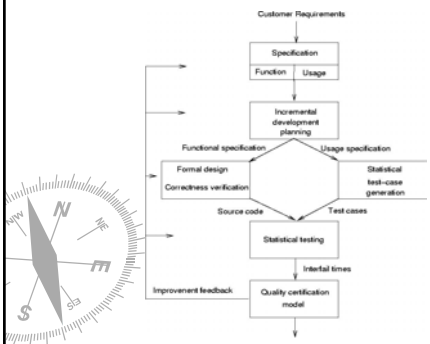
- Avoid dependence on costly defect-removal process by writing code increments ***RIGHT FOR THE FIRST TIME*** and verify their correctness before testing.
- The theoretical foundations of cleanroom are
  1. Formal specifications and design
  2. Correctness verification
  3. Statistical testing.

Notice: bug removal is absent, because they are not supposed to appear in Cleanroom.

Cleanroom:

- Team correctness verification takes the place of unit testing and debugging. Software enters system testing directly, with no execution by development team. All errors are accounted for from first execution on, with no private debugging permitted.

Cleanroom process chart



Quality Comparison

- Comparison is meaningful only at the time of first execution.
- Traditional software exhibit 25-35 or more errors per KLOC at the time of unit testing.
- Cleanroom process demonstrated 2.3 errors per KLOC in million lines of code. This number represent all the errors measured from first ever execution through test completion.

...Quality Comparison

- Errors left behind by Cleanroom correctness verification are not because of complex design or interface errors. They are simple errors which are easy to find and fix by statistical testing.

Cleanroom: An incremental approach

- The cleanroom process enforces developing and certifying a pipeline of software increments that accumulates into the final system.
- The increments are developed, and certified by small, independent teams, with teams of teams for large projects.

Incremental approach …

- System integration is continual and functions grows with the addition of successive increments.
- Any future increment are predefined by the existing system. It minimizes the interface and design errors.
- Develop high quality software with "Quick and Clean" approach and new version is released which incorporates new requirements from users experience.

Cleanroom: Requirements are paramount.

- A cleanroom team(s) analyzes and clarifies customer requirements to the minute details after lots of interactions and feedback.
- In case of doubt, a team can develop a rapid prototype to get more feedback from the users.

Cleanroom process activities

Cleanroom development involves two cooperating teams and five major activities.

- Specification
- Incremental planning
- Design and verification
- Quality certification
- Feedback

Specification

Functional Specification:

- Developed by development team.
- Defines the required external system behavior in all circumstances of use.
- It is the basis for incremental software development.

Usage Specification:

- Developed by certification team.
- Defines all possible usage scenarios and their probabilities both correct and incorrect.
- It is the basis for generating test cases for statistical testing and quality certification.

Incremental Planning

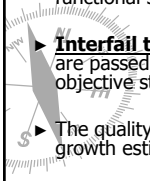
- On the basis of specifications, both development and certification teams define initial plans for developing increments.
- Break the project into small pieces according to their sizes and complexities.
- Parallel developments are allowed.

Design and verification

- The development team carries out design and correctness verification cycle for each increment.
- The certification team work in parallel, using the usage specification to generate test cases that reflect on the expected use of accumulating increments.

Quality Certification

- Periodically, the development team integrates a completed increment with prior increments and hand-over to the certification team.
- Certification team executes the entire software using statistical test cases. The results are checked against the functional specification.
- **Interfail time**, i.e. the elapsed time between failures, are passed to a quality certification model that computes objective statistical measures of quality.
- The quality certification model employs a reliability growth estimator to derive statistical measures of quality.



Quality Certification

- Contrary to other models, cleanroom certifications are done continuously, over the life of the project.
- Higher level increments enters the certification first and therefore major architectural and design decisions are validated in execution before development teams work on them.
- Since certification is done for all increments as they accumulate, higher level are subjected to more testing than the lower level increments.

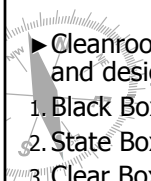
Feedback

- Certification team passes errors to the development team. If the quality is low then the managers and team initiate process improvement.
- At every level of increment, we have assured quality and intellectual control.

Functional Specification:

- The cleanroom process is built upon function theory where each program is treated as rules for mathematical functions subject to stepwise refinement and verification.
- Cleanroom uses Box Structure specification and design model.

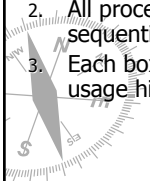
1. Black Box
2. State Box
3. Clear Box



Box Structure

Three principles that govern use of box structure

1. All data defined in a design is encapsulated in boxes.
2. All processing is defined by using boxes sequentially or concurrently.
3. Each box occupies a distinct place in a system's usage hierarchy.



Black Box:

- A black box is a precise specification of external, user-visible behavior in all possible circumstances of its use.
- The black box transition function

  {S,SH} -→ {R}

  Where {S} is set of all possible stimulus
  {SH} A set of stimulus history
  {R} A set of response which depends on {S} and {SH}

…Black Box

- The objective of black-box specification is to define the responses produced for every possible stimulus and stimulus history, including erroneous and unexpected stimuli.
- By defining behavior in terms of stimulus histories, black-box specifications neither depend on nor prematurely define design internals.

…Black Box

Stimulas	Conditions
S1	C1
S2	C2
S3	C3
Sn	Cn

To record large specifications, classes of behavior are grouped into nested tables.

State Box

- Transition function

  {S,OS}-→ {R,NS}

  Where OS : Old State NS : New State

- This state is same as Black Box, but all the histories are referenced by old state.
- From the OO point of view, this state encapsulate state data and services on that data.

Clear Box

- The transition function is

  {S,OS}→{R,NS} by procedure

- A clear box is simply a program that implements the corresponding state box.
- A clear box may invoke black boxes.
- A clear box allows defining new objects or extensions to existing ones.

Box Verifications

- Each state box is verified with respect to their black boxes and clear boxes with respect to their state boxes. Box structure bring correctness verification to object architecture.

Correctness Verification

- Basic control structures in programming are sequence, alternation (if-then-else) and iterations (while-do)
- Every control structure is verified in turn by all the teams. The requirement for unanimous agreement results in software that has fewer or no defects.
- Functional verification is superior than unit testing, because unit testing checks only a few test paths out of many possible paths, but a function checking requires only one check.
- Most functional verifications conditions can be checked in a few minutes, but unit tests take substantial time to prepare, execute and check.

Quality Certification

- Statistical approach is adopted as there are too many tests to perform.

  How long, on an average, will a software product execute before it fails ?

1. **Sampling:** All possible executions (correct and incorrect)
2. **Measurement:** Measure the quality by determining if the executions are correct.
3. **Extrapolation:** Extrapolate the quality of sample to the population of possible execution.
4. **Correction:** Identify and correct flaws in the development process.

…Quality Certification…

- User Centric: Statistical usage testing focuses on the way users intend to use the software.
- It is external behavior testing, not the internals of design and implementation.
- Cleanroom testers should have good knowledge about the usage of the software and not how it was developed.

Usage Probability Distribution

Focus on usage of the system. It defines.

- All usage patterns and conditions (correct and incorrect) with the probabilities of occurrence.
- There sources are specifications, interviews with prospective users and the pattern of uses in the previous version.
- For large-systems, UPD are often recorded in formal grammars or Markov chains for analysis and automatic processing.
- Certain low-probability but high-risk failure can be given higher importance.

Test case generation

- Test cases are generated from UPD which simulates users experience with the product. Since it requires only UPD, producing them is mechanical and automation is possible.
- For large systems, usage grammar or Markov chains can be used to generate test cases.

Statistical testing

- High-rate errors are responsible for nearly 2/3 of software failures even though they comprise less than 3% of total errors.

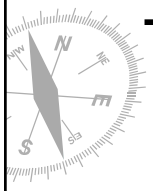
  Because statistical testing amount to testing software the way users will use it, high-rate errors tend to be found first.
- Traditional coverage testing find errors in random order, it is less effective than statistical testing.

Success Stories

Project	Resource	Size	CleanRoom Quality	Additional
Martin Marietta Automated Documentation System	Four Person	1820 lines	0.0	
IBM AOEXPERT/MVS	50 Person	107 KLOC PL/I, Rexx	2.6E/KLOC	Productivity 486 LOC/month
NASA Satellite Control		40 KLOC Fortran	4.5E/KLOC	Productivity 780 LOC/month
Ericsson Telecom	70 person	350 KLOC Operating system	1.0 E/KLOC	Increased by 70%
IBM Cobol/SF	Six person team	85 KLOC PL-I	3.4 E/KLOC	

Notice: The success stories have not mentioned how many errors were present in the original system.

The End



Thank You

It is good to be disciplined but …

- Is cleanroom suitable for a new project with changing requirements ?
- How much it cost to develop a software by cleanroom process compared to other models ?
- Does language issue play an important role in Cleanroom process ?
  Compare FORTRAN v/s JAVA.
- Is it practical for small scale software company to invest time and money whose survival depends on days progress than years.
- Despite lots of errors, Microsoft is still a saleable product.

Need Classification for Bugs

- Is bug an over-abused word ? How do we differentiate software malfunctions created by ignorance, mistake or intensions ?
  Which one Cleanroom solves best ?

Do these cause bugs or errors ?

1. Simplified Assumptions
2. Increasing Complexities of machines
3. Lack of standards
4. Using old horses for year 2003 rugby
5. Optimization for hardware