정형기법

한국항공대학교 소프트웨어학과 지승도교수
R.S. Pressman

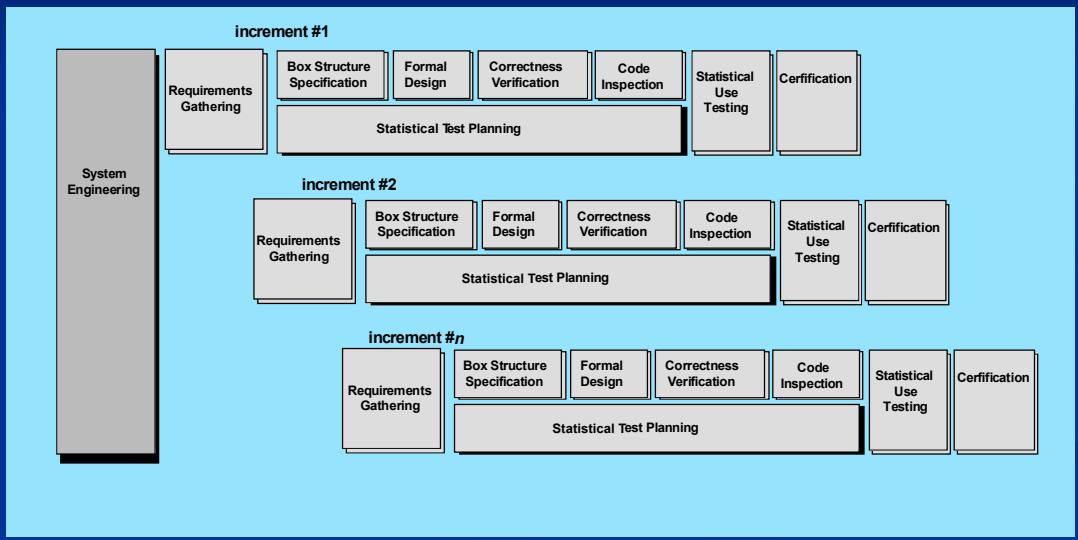
Cleanroom Software Engineering

Approach that emphasizes the need to build correctness into software. Instead of the classic analysis, design, code, test and debug cycle, the cleanroom approach suggests a different point of view.

➔ called “Box structure specification”

➔ ‘Box’ encapsulates the system.

The Cleanroom Process Model



The Cleanroom Strategy-I

Increment Planning—adopts the incremental strategy

Requirements Gathering — defines a description of customer level requirements (for each increment)

Box Structure Specification — describes the functional specification

Formal Design — specifications (called "black boxes") are iteratively refined (with an increment) to become analogous to architectural and procedural designs (called "state boxes" and "clear boxes," respectively).

Correctness Verification — verification begins with the highest level box structure (specification) and moves toward design detail and code using a set of "correctness questions." If these do not demonstrate that the specification is correct, more formal (mathematical) methods for verification are used.

Code Generation, Inspection and Verification — the box structure specifications, represented in a specialized language, are transmitted into the appropriate programming language.

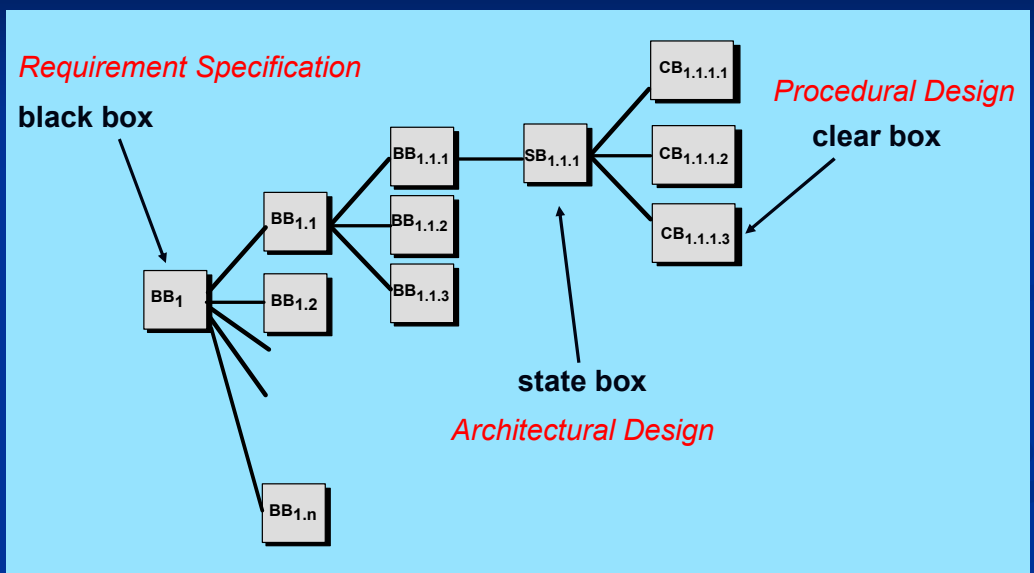
The Cleanroom Strategy–II

Statistical Test Planning — a suite of test cases that exercise of "probability distribution" of usage are planned and designed

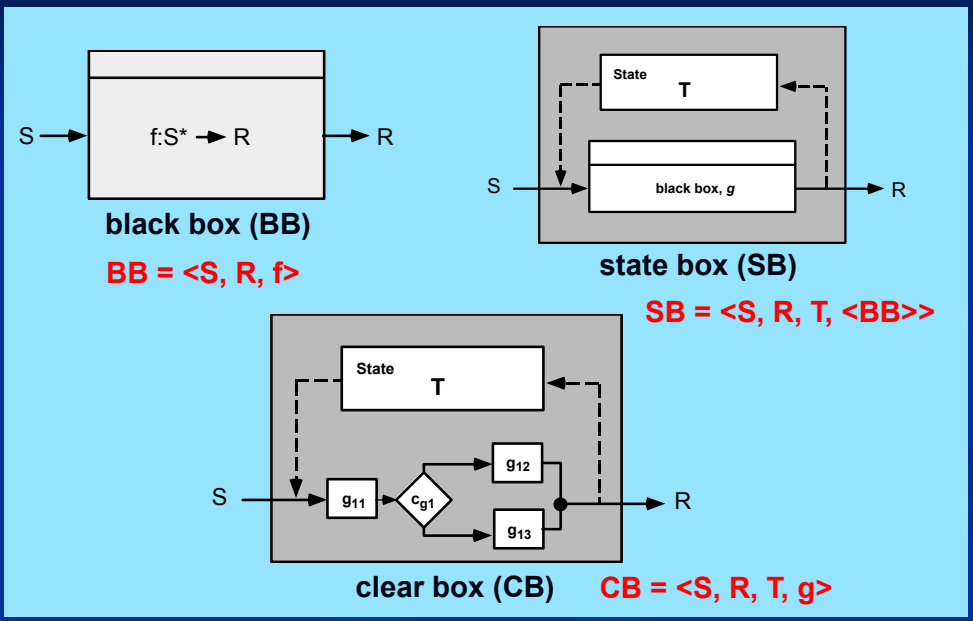
Statistical Usage Testing — execute a series of tests derived from a statistical sample (the probability distribution noted above) of all possible program executions by all users from a targeted population

Certification — once verification, inspection and usage testing have been completed (and all errors are corrected) the increment is certified as ready for integration.

Box Structure Specification



Box Structures



Problems with Conventional Specification

- contradictions
- ambiguities
- incompleteness
- mixed levels of abstraction

Formal Methods

- Mathematically based technology for describing system properties — consistency, completeness, and lack of ambiguity
- The formal syntax of a specification language enables requirements or design to be interpreted in only one way, eliminating ambiguity that often occurs when a natural language (e.g., English) or a graphical notation must be interpreted

Formal Specification Language 예

- The block handler
  - The block handler maintains a reservoir of unused blocks and will also keep track of blocks that are currently in use. When blocks are released from a deleted file they are normally added to a queue of blocks waiting to be added to the reservoir of unused blocks.
    - The state

      used, free: ℙ *BLOCKS*

      BlockQueue: *seq* ℙ *BLOCKS*
    - Data Invariant

      used ⩤ *free* = \

      used ⩥ *free* = *AllBlocks*

      i: dom *BlockQueue* *BlockQueue i* # *used*

      i, j : dom *BlockQueue* *i* ≠ *j* => *BlockQueue i* ⩤ *BlockQueue j* = \
    - Precondition

      #BlockQueue > 0
    - Postcondition

      used' = *used* ⩥ *head BlockQueue*

      free' = *free* ⩤ *head BlockQueue*

      BlockQueue' = *tail BlockQueue*