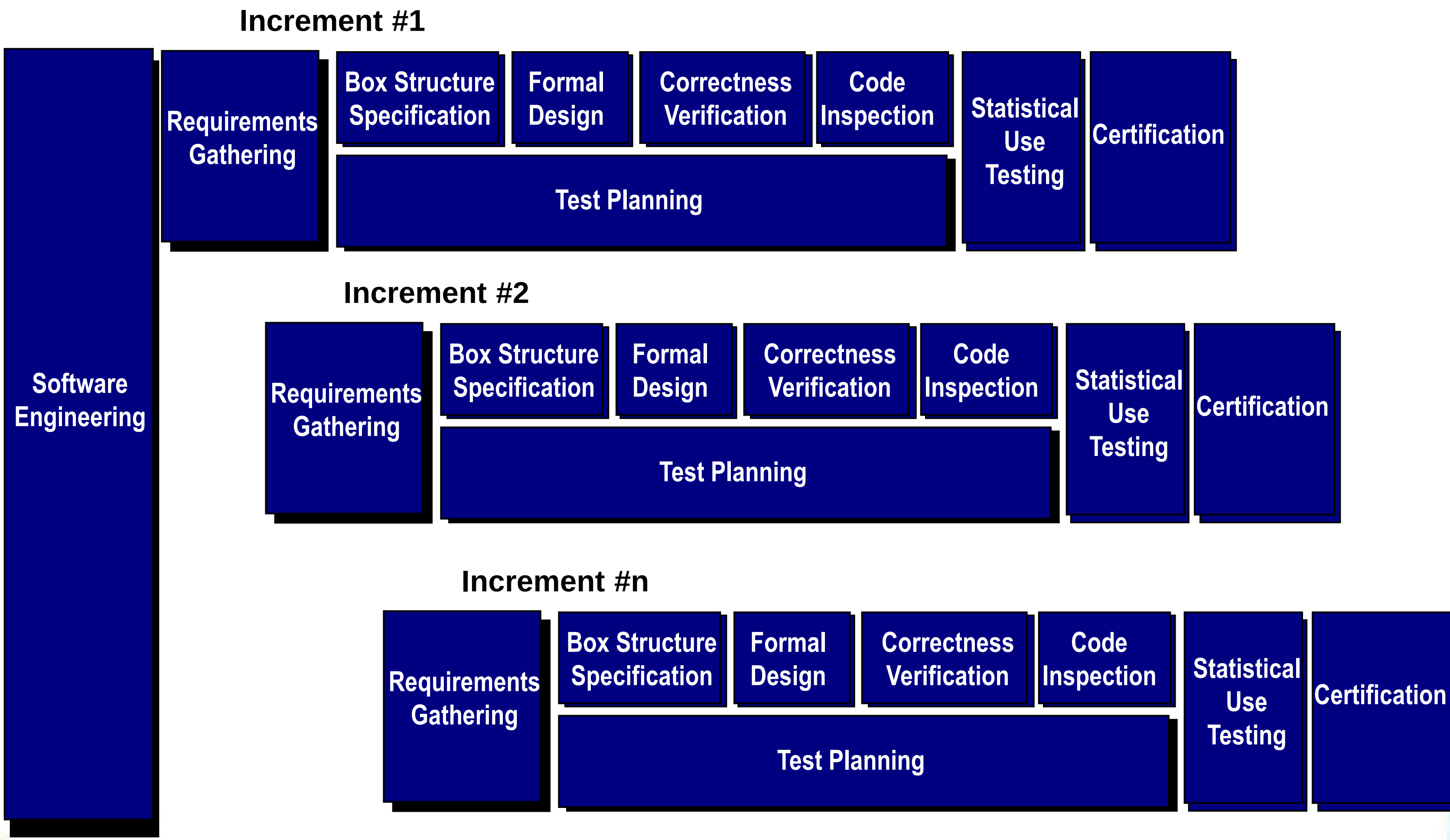
Ch.28 Formal Modeling and Verification

Formal Modeling and Verification

- **Cleanroom software engineering and formal methods**
  - Both demand a specialized *specification* approach and each applies a unique *verification* method.
  - Both are quite *rigorous* and neither is used widely by the software engineering community.
- **For "bullet-proof" software**

28.1 The Cleanroom Strategy



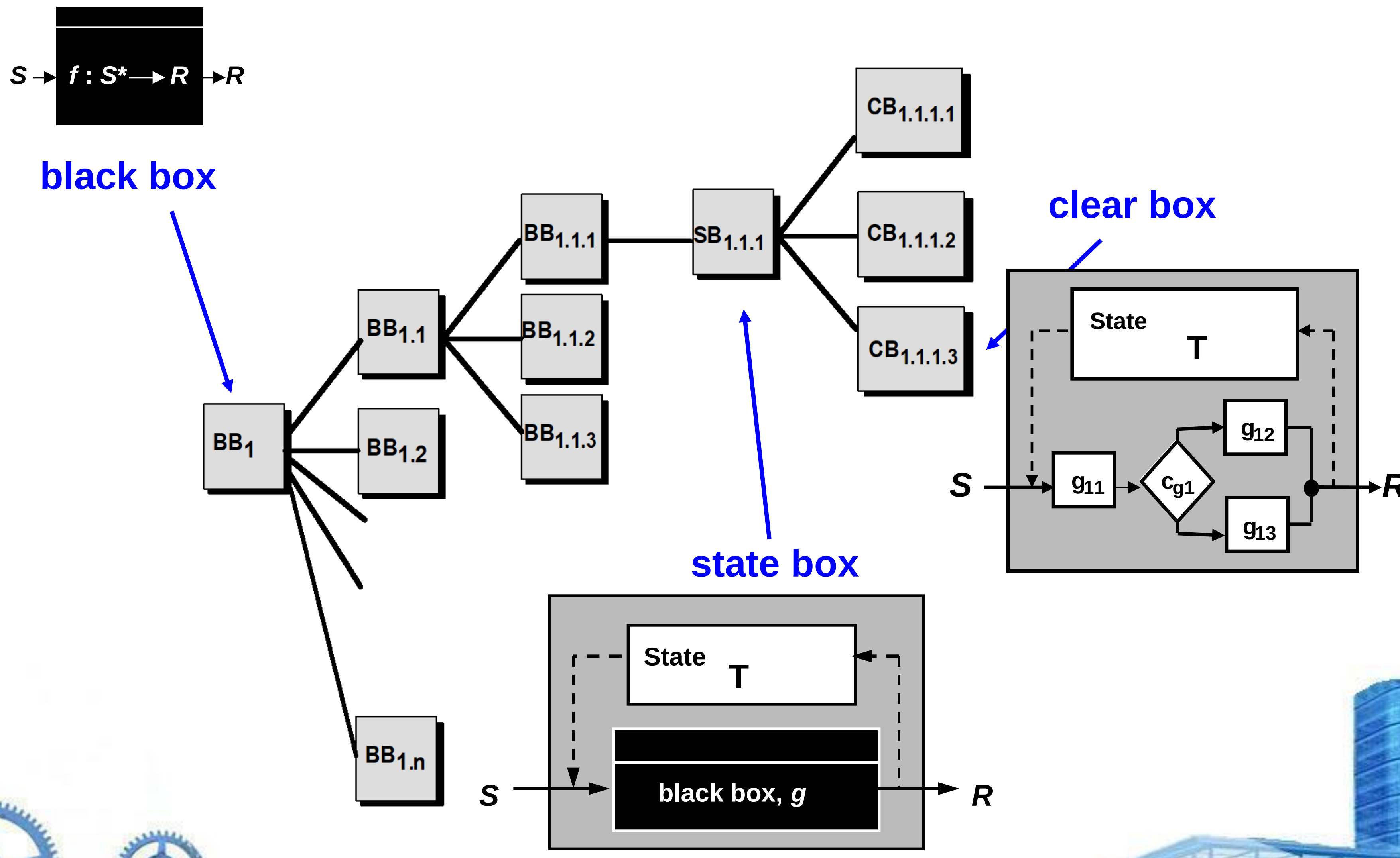
28.1 The Cleanroom Strategy

- **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

28.1 The Cleanroom Strategy

- **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.

28.2 Functional Specification



28.3 Cleanroom Design

- **Design Refinement**
  - If a function *f* is expanded into a sequence *g* and *h*, the correctness condition for all input to *f* is:
    - **Does *g* followed by *h* do *f* ?**
  - When a function *f* is refined into a conditional (if-then-else), the correctness condition for all input to *f* is:
    - **Whenever condition <c> is true does *g* do *f* and whenever <c> is false, does *h* do *f* ?**
  - When function *f* is refined as a loop, the correctness conditions for all input to *f* is:
    - **Is termination guaranteed?**
    - **Whenever <c> is true does *g* followed by *f* do *f*, and whenever <c> is false, does skipping the loop still do *f* ?**

28.3 Cleanroom Design

- **Design Verification**
  - It reduces verification to a finite process
  - It lets cleanroom teams verify every line of design and code
  - It results in a near zero defect level
  - It scales up
  - It produces better code than unit testing

28.4 Cleanroom Testing

- **Statistical Use Testing**
  - tests the actual usage of the program
  - determine a “**usage probability distribution**”
    - analyze the specification to identify a set of stimuli
    - stimuli cause software to change behavior
    - create usage scenarios
    - assign probability of use to each stimuli
    - test cases are generated for each stimuli according to the usage probability distribution

28.4 Cleanroom Testing

- **Certification**

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

28.4 Cleanroom Testing

- **Certification**
  - **Sampling model.** Software testing executes m random test cases and is certified if no failures or a specified numbers of failures occur. The value of m is derived mathematically to ensure that required reliability is achieved
  - **Component model.** A system composed of n components is to be certified. The component model enables the analyst to determine the probability that component i will fail prior to completion
  - **Certification model.** The overall reliability of the system is projected and certified

28.5-6 Formal Methods

Formal methods used in developing computer systems are mathematically based techniques for describing system properties. Such formal methods provide frameworks within which people can specify, develop, and verify systems in a systematic, rather than ad hoc manner.

—— *The Encyclopedia of Software Engineering* [Mar01]

- **The Problem with conventional specs**
  - contradictions
  - ambiguities
  - vagueness
  - incompleteness
  - mixed levels of abstraction

28.5-6 Formal Methods

- **Formal Specification**
  - Desired properties— ***consistency, completeness, and lack of ambiguity*** —are the objectives of all specification methods
  - 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
    - The descriptive facilities of set theory and logic notation enable clear statement of facts (requirements)
  - ***Consistency*** is ensured by ***mathematically proving*** that initial facts can be formally mapped (using inference rules) into later statements within the specification

28.5-6 Formal Methods

- **Concepts**
  - **data invariant**— a condition that is true throughout the execution of the system that contains a collection of data
  - **state**
    - Many formal languages, such as OCL (Appendix3) , use the notion of states, that is, a system can be in one of several states, each representing an externally observable mode of behavior
    - The Z language (Appendix3) defines a state as the stored data which a system accesses and alters
  - **operation**— an action that takes place in a system and reads or writes data to a state
    - **precondition** defines the circumstances in which a particular operation is valid
    - **postcondition** defines what happens when an operation has completed its action