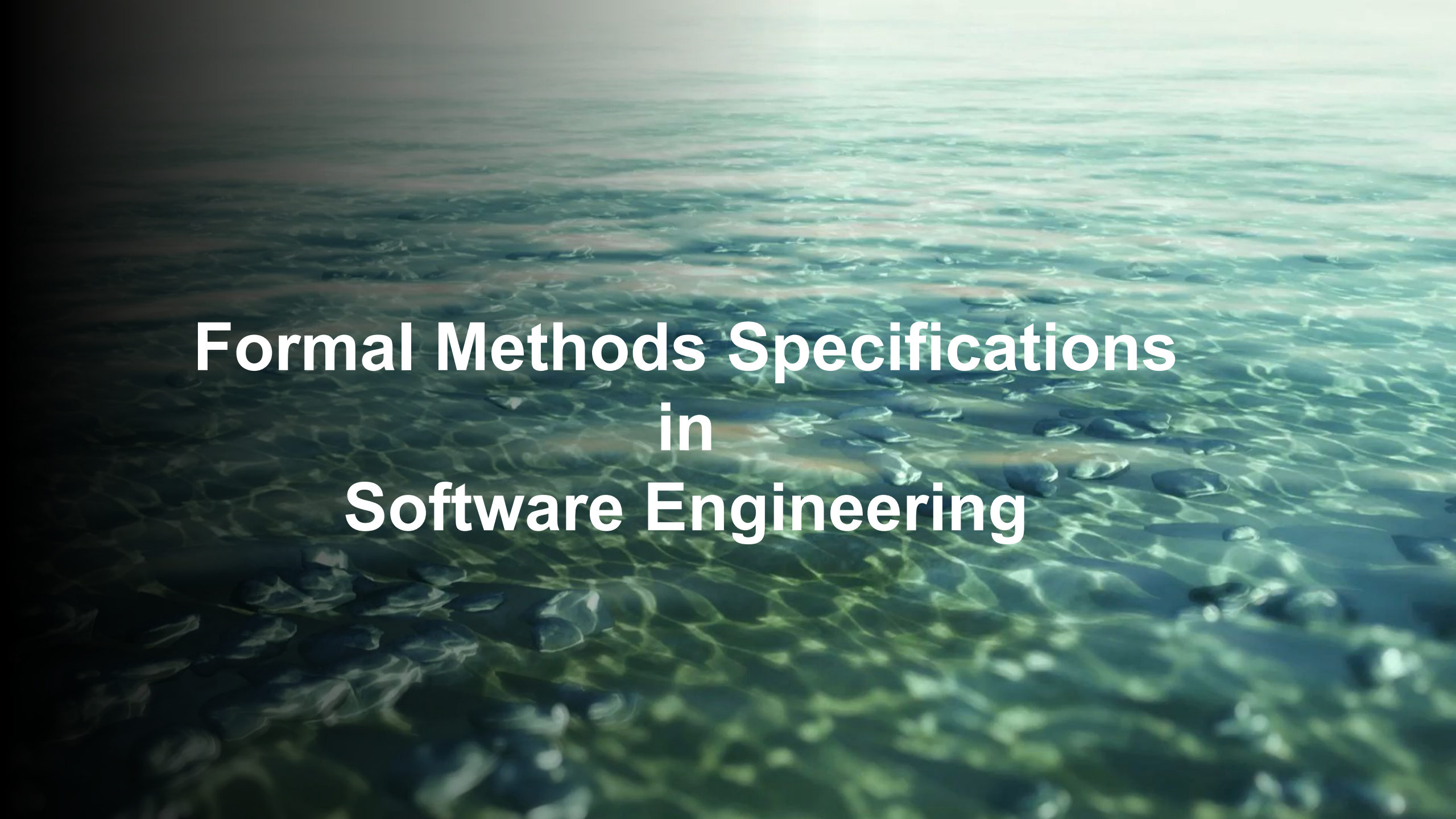

Formal Methods Specifications
in
Software Engineering

Week-1: Introduction and Objectives

To Do-------

- What are formal Methods?
- Why formal specifications Methods?
- Formal Specification Methods and Software Engineering
- Formal Methods Can be practical
- Types of Formal specifications. Formal Proofs and Mathematical Models
- Characteristics of formal specifications and Purpose.
- Formal Specifications Process Models
- Cleanroom software Development and Process
- Formal Methods can be practical

What are formal Methods?

Challenge of Software Engineering:

The main challenge in software engineering is to deliver *high-quality software on time* and *on budget to customers.*

What is Quality?

During development process, there is a continuous assessment and measurement to determine conformance to the following requirements:

- *measurement model*
- *project tracking and oversight*
- *validation criteria*
- *quality assurance system*
- *plans, commitment to improvement*

The use of process models is encouraged

Requirements must be clearly stated such that they cannot be misunderstood:

Managerial concerns

- *complete*
- *unambiguous*
- *verifiable*
- *precise*
- *concise*
- *consistent*

Technical concerns

The use of formal methods is encouraged

What are formal Methods?

Definitions:

A mathematical notation consisting of well defined syntax and semantics used to unambiguously specify the requirements and modelling of a software system.

A **formal method** is used to establish proof of correctness of the final implementation of software systems with respect to its specifications.

Formal Methods can also be described as

- Mathematical Techniques and tools based on mathematics and formal logic to do requirements specifications and verifications, and development of software
- Assuming various forms and levels of rigor
  - Informal
  - Low
  - Medium
  - High

Why Formal Specifications?

The development of a formal specification provides insights and an understanding of the software requirements and software design. Formal Specifications should

- Add clarity and understanding by giving a description of the system that is
  - Complete
  - Unambiguous
  - Easily analyzed
- Lead to a better code tht is
  - Reliable
  - Accurate
  - Maintainable
  - Reusable
  - Veriable

Definition[Formally]

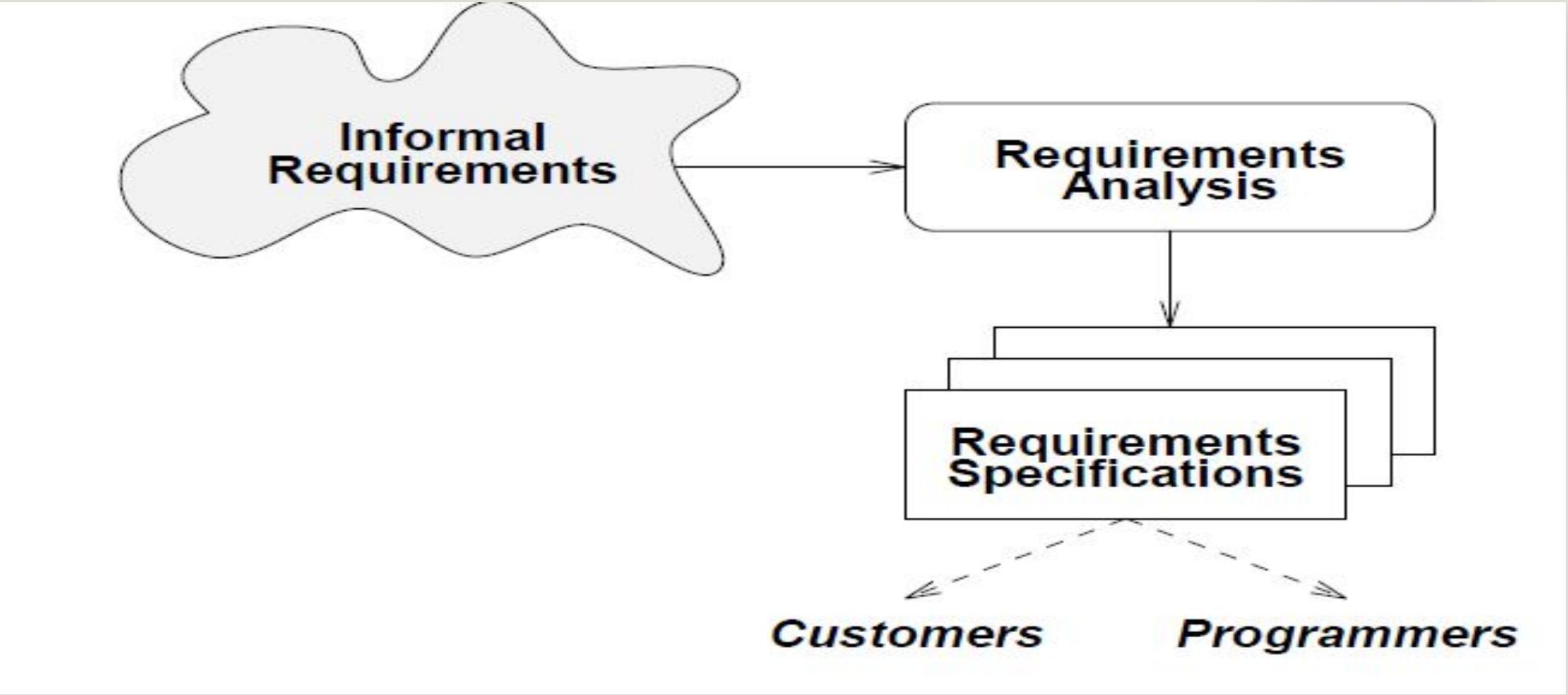
Formal specification is the use of mathematical notation to describe in a precise way the properties that an information system must have, without unduly constraining the way in which these properties are achieved.

When to Use Formal Methods

Software Development Life Cycle:

- Requirements analysis
- Requirements specification
  - As a process: when the functionality of the software is specified
  - As a product: where the expected functionality is recorded
- Architectural design
- Detailed design
- Implementation
- Testing

Requirements Specification as a Contract



Source:*EECS810: Software Engineering (University of Kansas, Fall 2006)*

Formal Specification Methods and Software Engineering

Formal Specifications:

- Is not a replacement, but an enhancement of the existing technologies
- Can only be effective if integrated within an overall methodology for software engineering
- <u>Implication of Using Formal Specifications</u>
  - Training in the use of notations
  - Integration with informal methodologies
  - Translation or client consumption
  - Emphasis on abstraction

Formal methods by themselves will not solve the software crisis

Assignments I

1.Describe in two pages Benefits, Drawbacks of Formal Methods

2.Find the papers “

i)J. Woodcock, P.G. Larsen, J. Bicarregui, J. Fitzgerald, Formal methods: practice and experience. ACM Comput. Surv. 29”

ii). J. Bowen and H. Hinchey (editors), *Applications of Formal Methods*, Prentice-Hall International, 1995.

Summarize into 2-page applicationsof formal methods.

Practical applications of Formal Methods

Applications of Formal methods exists in several areas of real world, namely

- Transportation sector
- Nuclear and Space sector,
- Defense sector,
- Semiconductor sector,
- Financial Institutions
- and the telecom sector.

The scope of the applications of formal methods include the followings

- Not only **formal specification**
- **specification with inspections**,
- proofs,
- refinement,
- test generation and
- model checking

How to Use Formal Methods

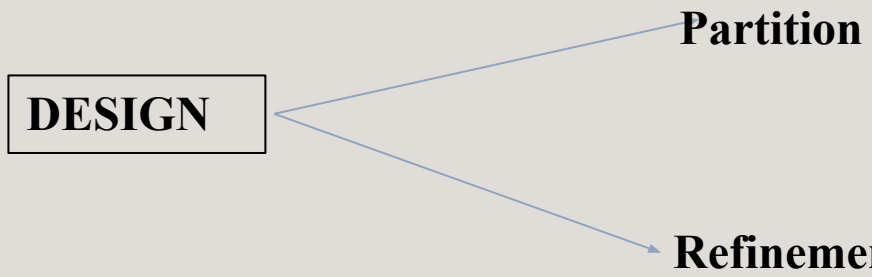
Formal Methods can be used in the following ways

1. Modelling: Models enable us to describe and predict program behavior. A mathematical model can be used to describe program behavior accurately and comprehensively, and it is often much shorter and clearer than the code. To be able to predict a program is an essential property of safety-critical systems.

Example:A prototype that demonstrates the **look and feel** of a new system to prospective users is a kind of model because it does not provide all of the functions of the final product. It is a bit like an architect's scale model or an artist's rendering of a new building, because it is intended to convey an aesthetic impression.

A mathematical model that represents the intended behavior of a program can be used as a *formal specification.*

2.**Design:**Design means organizing the internal structure of a program



Partition: This implies dividing the whole system into parts or *modules* that can be developed independently.

Refinement: This implies adding detail, going from an abstract model that clearly satisfies the original requirements to a concrete design that is closer to code.

Many informal software development methods address design. Most of them teach a particular way to draw and annotate diagrams that you can use to document designs, such as ***bubble-and-arrow data flow diagrams.***

Z is a more powerful design notation because it can also model behavior ***and can also*** support constructive approaches to design.

3.**Verification:** Verification means showing that our code will do what we intend.

Verification deals with the final product of our development: code in some executable programming language. One of the products of a formal verification is a ***proof*** which is described as a convincing demonstration — based only on the specification and the program text, not on ***executing*** the program — that the code does what its specification requires.

Proof can provide greater confidence than ***testing*** because it considers all cases, while testing just samples some of them. Moreover, proof can be more convincing than appeals to intuition because it can be more explicit, easier to check, and therefore less fallible than intuition.

Types of Formal specifications.

There are two basic views to formal methods:

Model-oriented approach:-The model-oriented approach to specification is based on mathematical models, where a model is a simplification or abstraction of the real world that contains only the essential details.

Example: The model of an aircraft will not include the color of the aircraft, and the objective may be to model the aerodynamics of the aircraft.

The model-oriented approach to software development involves defining an abstract model of the proposed software system, and the model is then explored to determine its suitability as a representation of the system.

Model-Oriented construct a model of the system behavior using mathematical objects like sets, sequences etc. The implementation techniques include

- Statecharts, SCR, VDM, Z
- Petri Nets, CCS, CSP, Automata theoretic models

Formal Specification Types[contd]

Axioms-based/Property-based Approach: This focuses on the properties that describe the behavior the proposed system is to satisfy such as axioms, rules and there is no intention to produce an abstract model of the system.
The required properties and behaviour of the system are stated in mathematical notation.

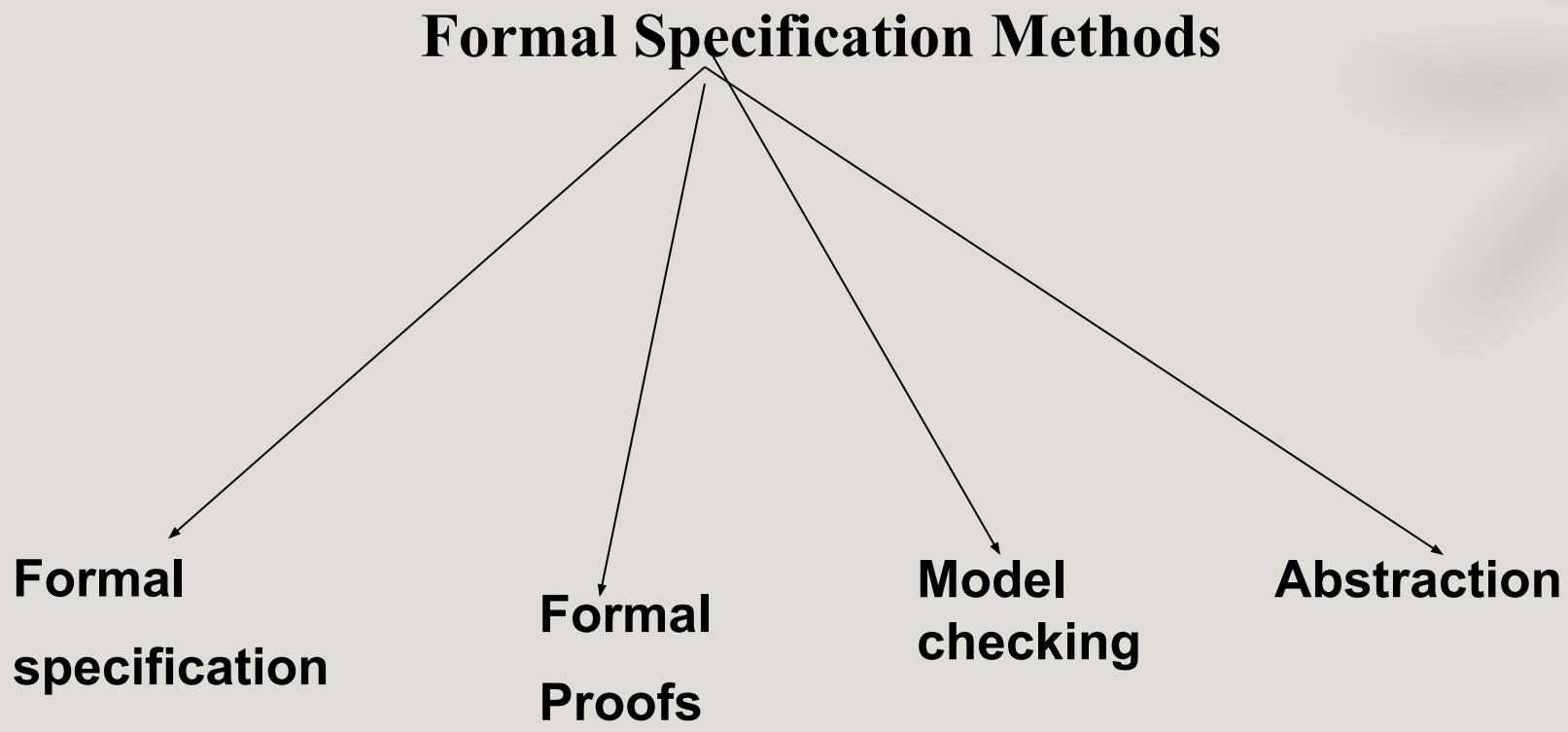
The difference between the axiomatic specification and a model-based approach may be seen in the example of a stack.

The stack includes operators for pushing an element onto the stack and popping an element from the stack. The properties of pop and push are explicitly defined in the axiomatic approach.

The model-oriented approach constructs an explicit model of the stack, and the operations are defined in terms of the effect that they have onthe model. The

The axiomatic specification of the pop operation on a stack is given by properties, for example,
pop(push(s, x)) = s.

Formal Methods Concepts



Formal Proofs

- Proof is an essential part of specification
- Proofs are constructed as a series of small steps, each of which is justified using a small set of rules
- Proofs can be done manually, but usually constructed with some automated assistance

Model Checking

- A technique relies on building a finite model of a system and checking that a desired property holds in that model
- Two general approaches
  - temporal model checking
  - automaton model checking
- Use model checkers
  - SMV

Abstraction

- Representation of the program using a smaller model
- Allows you to focus on the most important central properties and characteristics
- Getting the right level of abstraction is very important in a specification.

Mathematical Models

- Abstract representations of a system using mathematical entities and concepts
- Model should captures the essential characteristics of the system while ignoring irrelevant details
- Model can be analyzed using mathematical reasoning to prove system properties or derive new behaviors.
- Two types
  - Continuous models
  - Discrete models

Formal Specification Process Model

- Clarify requirements and high level design
- Articulate implicit assumptions
- Identify undocumented or unexpected assumptions
- Expose defects
- Identify exceptions
- Evaluate test coverage

Cleanroom software development

- *Cleanroom Software Engineering* software development process is an engineering process with ***mathematical foundations*** rather than a trial-and-error programming process. It consists of the followings
  - Spend a lot of effort "up-front" to prevent defects
  - Formal specification
  - Incremental development
  - Statistical methods to ensure reliability

- 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

Cleanroom Process Model

The Process Model can be described as follows

- 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.

Three Types of 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.

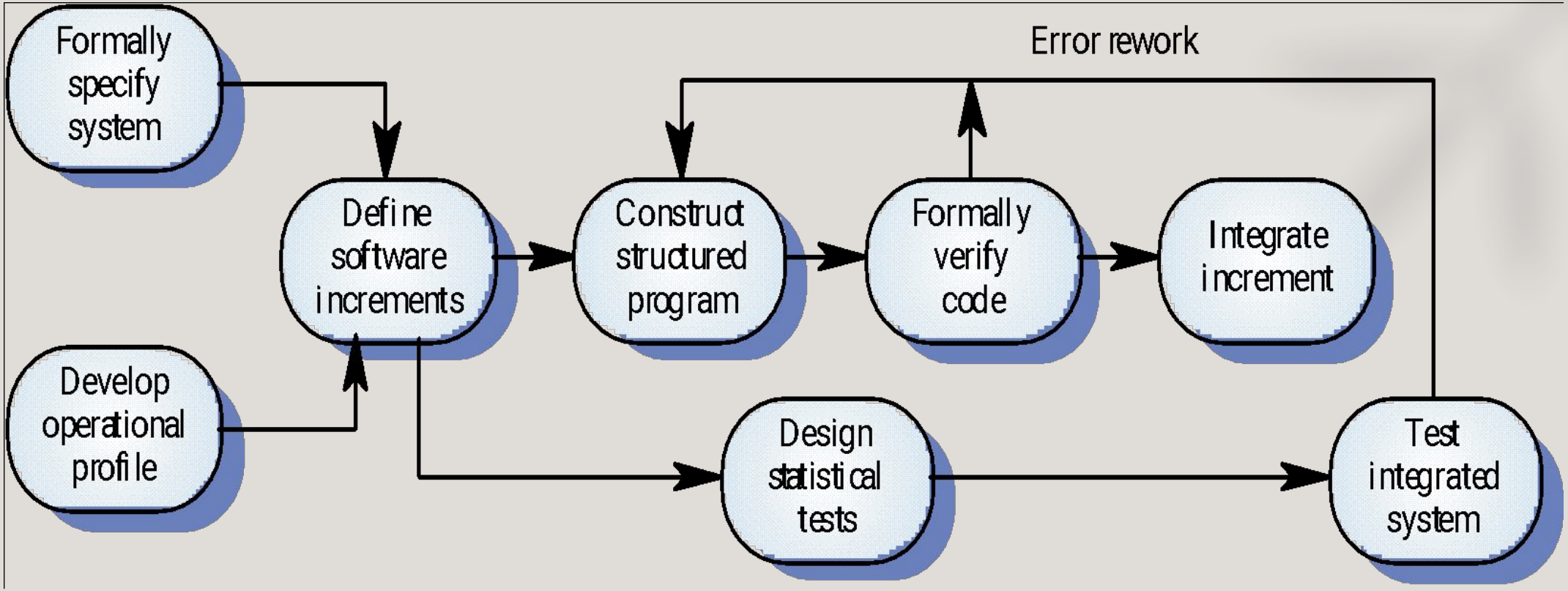
2. State box:

- 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 Process



Finite State Machine

A finite state machine (FSM) is an abstract mathematical machine that consists of a finite number of states. It includes a start state q_0 in which the machine is in initially; a finite set of states Q; an input alphabet R; a state transition function d; and a set of final accepting states F (where $F \subset Q$).

The ***state transition function*** takes the ***current state*** and an input and returns the ***next state***. That is, the transition function is of the form:

$$d : Q \times R \rightarrow Q.$$

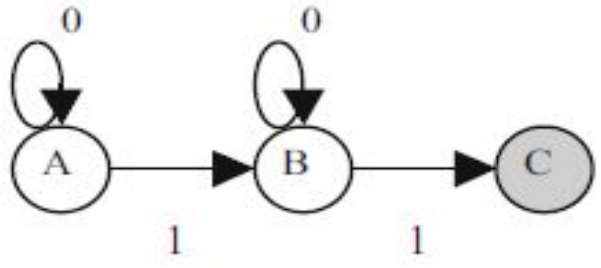
The ***transition function*** provides ***rules*** that define the action of the machine for each input, and it may be extended to provide output as well as a state transition

State Diagrams

State diagrams are used to represent finite-state machines, and each state accepts a finite number of inputs.

A finite-state machine may be ***deterministic*** or ***non-deterministic***, and a deterministic machine is represented as shown below

Deterministic changes to exactly one state for each input transition. whereas



Non-deterministic is a machine that may have a choice of states to move to for a particular input.

Finite-state automata can compute only very primitive functions and are not an adequate model for computing. There are more powerful automata such as the Turing machine that is essentially a finite automaton with an infinite storage (memory).

End of Week -1