MUST

Wisdom & Virtue

MIRPUR UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF SOFTWARE ENGINEERING

Software Design & Architecture

(Lecture # 4)

Software Design Concepts

Saba Zafar

(Lecturer)

Date: 8-11-2024

LECTURE CONTENTS

1. Basic Concepts of Software Design and Software Architecture

2. Software Development Activities

3. Software Design Representations

COURSE INFORMATION

Text Book Name: Software Engineering: A Practitioner’s Approach

Author: Roger S. Pressman, Bruce R. Maxim,

Edition: 8th or Higher

Reference Material:

1. Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, 2nd Ed, Universities Press, 2014.
2. Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures, Hassan Gomaa, Cambridge University Press, 2011.
3. Software Engineering, Sommerville I., 10th Edition, Pearson Inc., 2014

COURSE LEARNING OBJECTIVES

1. Describe and understand the role of design and its major activities within the Object Oriented software development process, with focus on the Unified process.
2. Design OOD models and refine them to reflect implementation details.
3. Apply and use UML to visualize and document the design of software systems.
4. Implement the design model using an object-oriented programming language.

ARCHITECTURE

ARCHITECTURE

IEEE DEFINITION

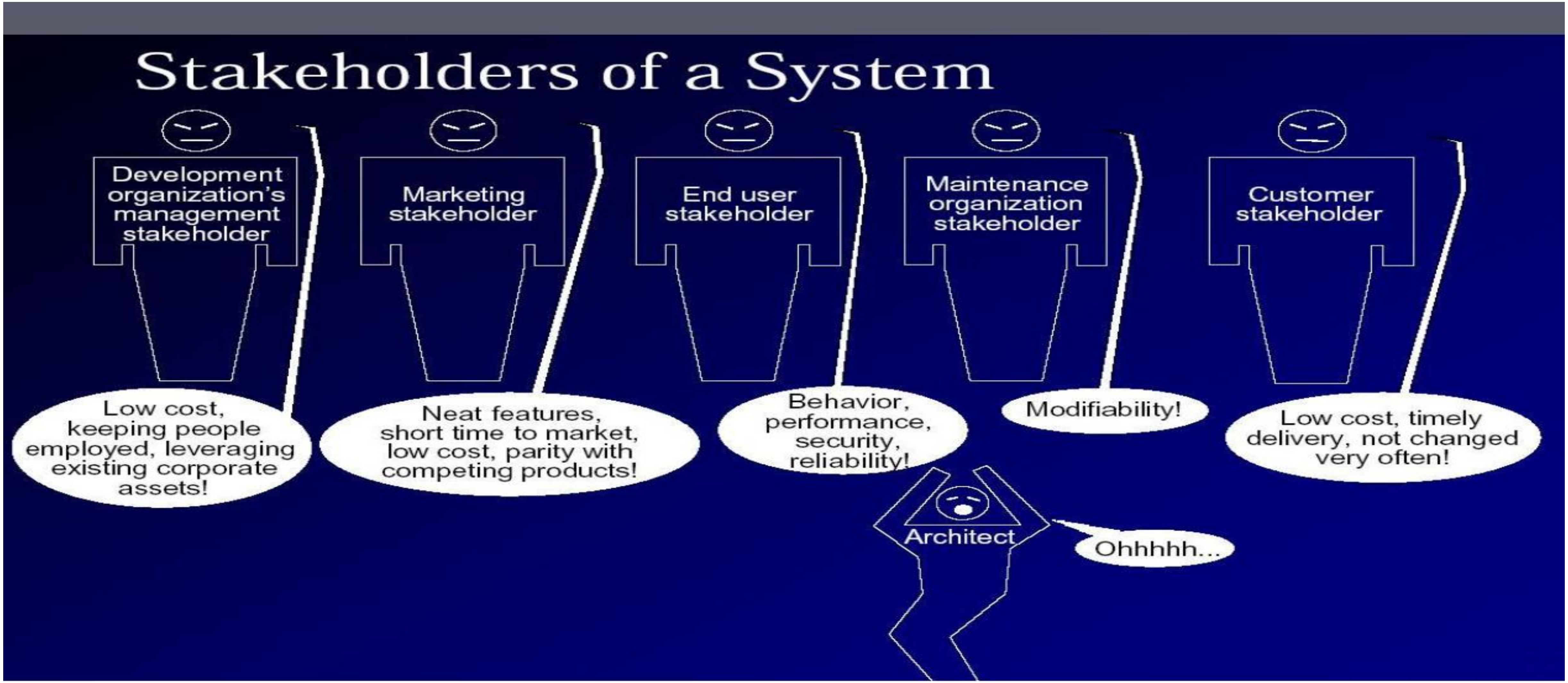
- *Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.*

SOFTWARE ARCHITECTURE

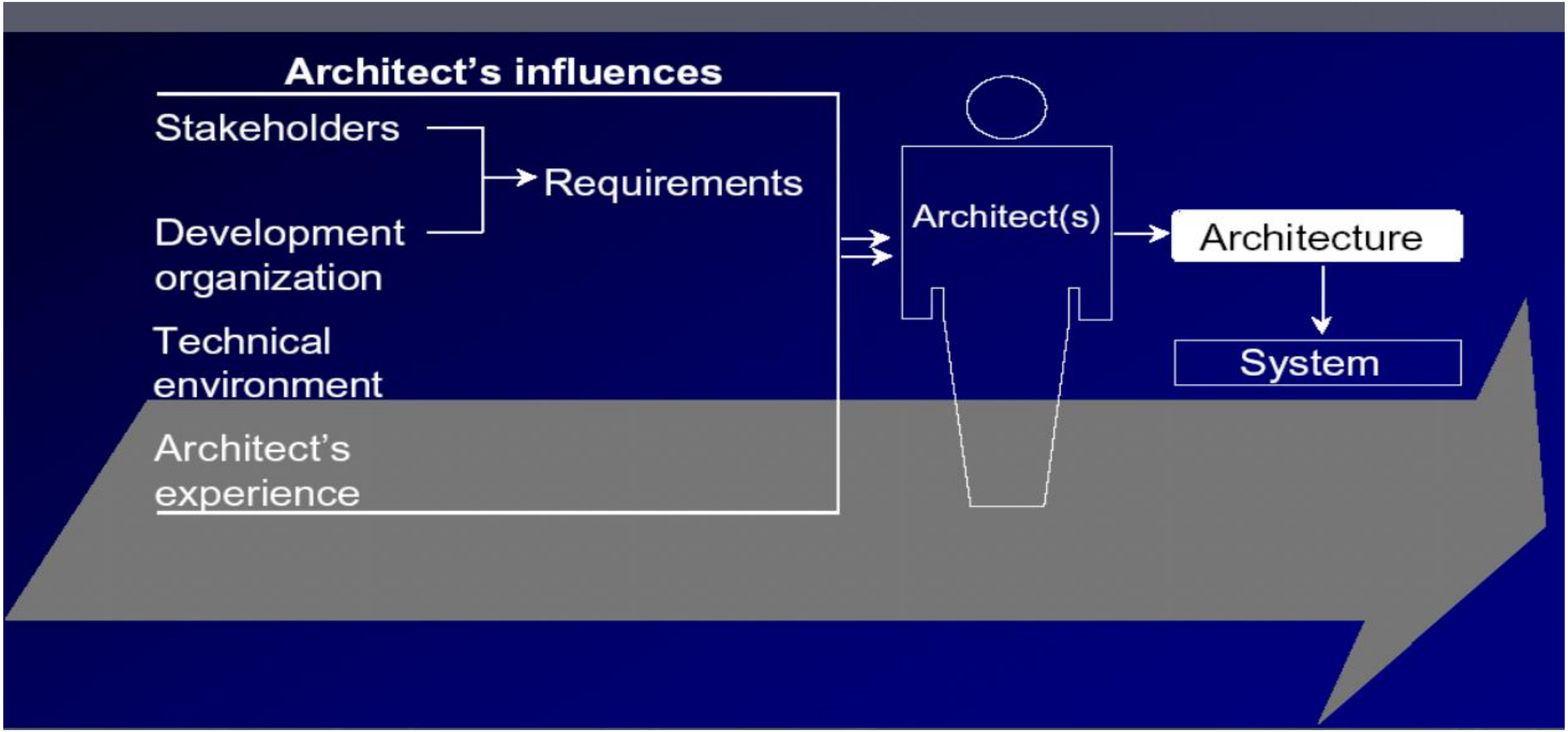
- *The software architecture of a program or computing system is the structure or structures of the system which comprise software elements, the externally visible properties of those elements, and the*

  Relationships among them.

WHO INFLUENCE SA?


Stakeholders of a System
Development organization's management stakeholder
Marketing stakeholder
End user stakeholder
Maintenance organization stakeholder
Customer stakeholder
Low cost, keeping people employed, leveraging existing corporate assets!
Neat features, short time to market, low cost, parity with competing products!
Behavior, performance, security, reliability!
Modifiability!
Low cost, timely delivery, not changed very often!
Architect
Ohhhhh...

SUMMARY : INFLUENCES ON THE ARCHITECT



WHY IS ARCHITECTURE IMPORTANT?

- *Handling Complexity*
- *Communication among stakeholders*
- *Early design decisions*
- *SA is a transferable ,reusable model*

WHY SOFTWARE DESIGN?

- *Engineered artifacts such as bridges, cars or television sets to be built without someone first designing them and construction plans*
- *Software is no different*
- *It occurs somewhere between the ‘Requirements Capture’ (where we decide what we would like our system to do) and ‘Implementation’ (where we build it).*

WHAT IS SOFTWARE DESIGN?

- ***Requirements*** *specification was about the WHAT the system will do*
- ***Design*** *is about the HOW the system will perform its functions*
  - *Provides the overall decomposition of the system*
  - *Allows to split the work among a team of developers*
  - *A software design is a meaningful engineering representation of some software product that is to be built*
  - *It is an abstraction of the software system*

SOFTWARE DEVELOPMENT ACTIVITIES

- ***Requirements Elicitation***
- ***Requirements Analysis*** *(e.g., Structured Analysis, OO Analysis)*
    - *Analyzing requirements and working towards a conceptual model without taking the target implementation technology into account*
    - *Part of requirements engineering*
- ***Design***
    - *Coming up with solution models taking the target implementation technology into account*
- ***Implementation***
- ***Test***

LECTURE CONTENTS

1. The Design Process
2. Stages of Design
3. Software Design Principles
4. Software Design vs Software Architecture

THE DESIGN PROCESS

- *Any design may be modelled as a directed graph made up of entities with attributes which participate in relationships.*
- *The system should be described at several different levels of abstraction.*
- *Design takes place in overlapping stages. It is artificial to separate it into distinct phases but some separation is usually necessary.*

DESIGN PROCESS

- *During the design process, the software specifications are transformed into design models that describe the details of the:*
  - *Data structures*
  - *System architecture*
  - *Interface*
  - *Components*
- *The emphasis in design phase/process is on quality*
- *This phase provides us with representation of software that can be assessed for quality*

STAGES OF DESIGN

- ***Problem Understanding* :** *Look at the problem from different angles to discover the design requirements.*
- ***Identify One or more solutions*:** *Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources.*
- ***Describe solution abstraction* :***Use Graphical, or other descriptive notations to describe the component of the design.*
- ***Repeat process for each identified abstraction*:** *until the design is expressed in primitive terms*

LEVELS OF SOFTWARE DESIGN

- ***Architectural design*** (*high-level design*)
  - *Architecture - the overall structure, main modules and their connections*
  - *It addresses the main non-functional requirements (e.g., reliability, performance)*
- ***Detailed design*** (*low-level design*)
  - *The inner structure of the main modules*
  - *Detailed enough to be implemented in the programming language*

SOFTWARE DESIGN SHOULD BE:

- *Simple*
- *Correct & Complete*
- *Loosely coupled*
- *Understandable*
- *Adaptable*

SOFTWARE DESIGN PRINCIPLES

- *The design process should not suffer from tunnel vision – A good designer should consider alternative approaches*
- *The design should be traceable to the analysis model*
- *The design should not reinvent the wheel*
- *The design should minimise intellectual distance b/w the software and the problem as it exists in the real world*
- *The design should exhibit uniformity and integration*
  - *A design is **uniform** if it appears that one person developed the whole thing*
  - *A design is **integrated** if care is taken in defining interfaces between design components*

SOFTWARE DESIGN PRINCIPLES(CONTD…)

- *The design should be structured to accommodate unusual circumstances, and if not, then it must terminate processing, do so in a graceful manner*
- *The design should be reviewed to minimize conceptual errors*
- *Design is not coding, coding is not design*
  - *Even when detailed designs are created for program components, the level of abstraction of the design model is higher than source code.*
- *The design should be structured to accommodate change*

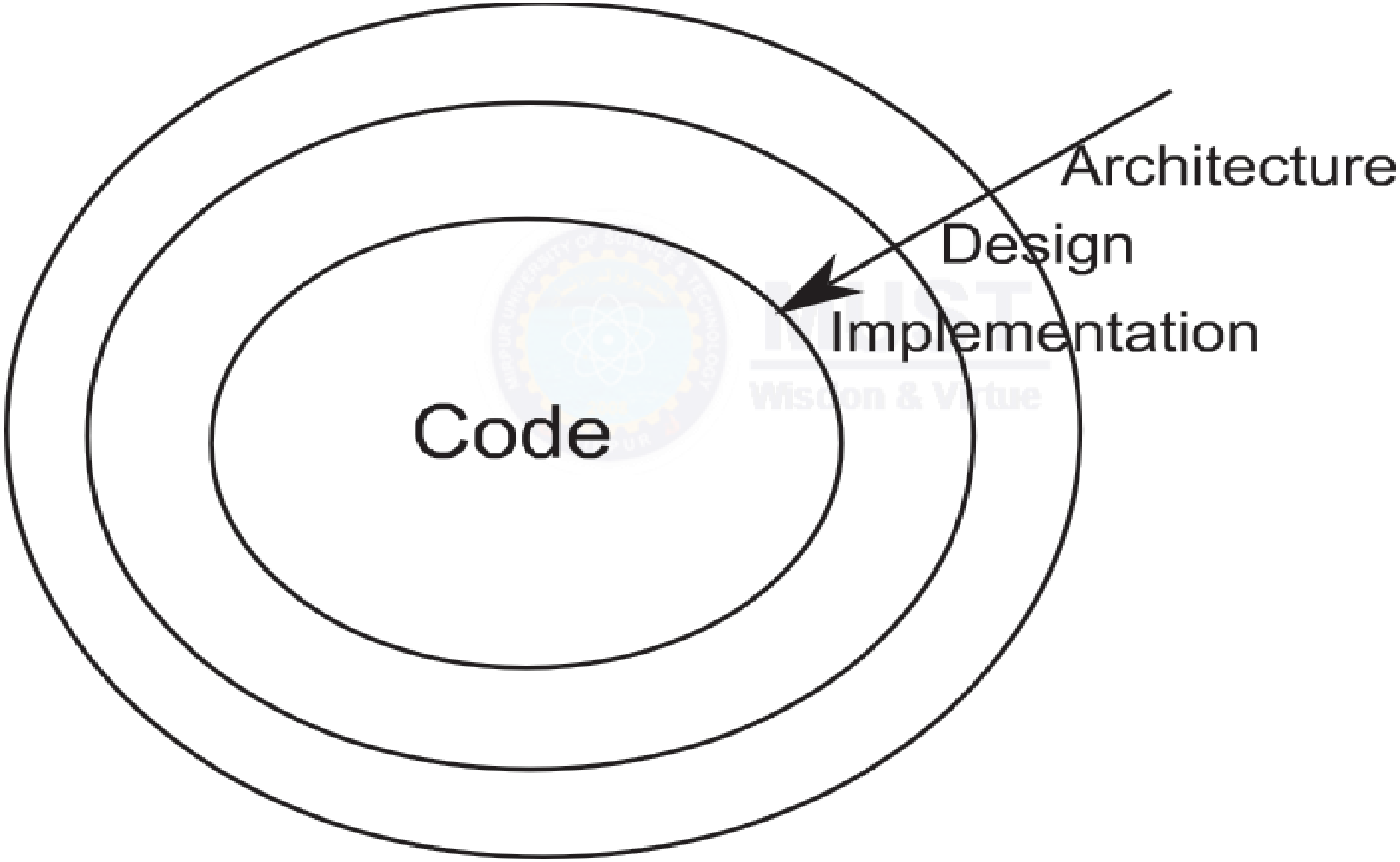
ARCHITECTURE VS DESIGN

- ***Architecture*** *is concerned with the selection of architectural elements, their interaction, and the constraints on those elements and their interactions*
- ***Design*** *is concerned with the modularization and detailed interfaces of the design elements, their algorithms and procedures*
- *Software architecture is "concerned with issues...beyond the algorithms and data structures of the computation."*
- *Architecture…is specifically not about…details of implementations (e.g., algorithms and data structures.)*

ARCHITECTURE VS DESIGN I

- *All architecture is design, not all design is architecture"*
- *Architectural design is outward looking*
  - *Focus on stakeholders, not technology*
- *Architecture doesn't describe the complete characteristics of components – Design does.*

ARCHITECTURE VS DESIGN II



LECTURE CONTENTS

1. Software Design Methods
2. Software Design Strategies
3. Concepts in Software Design Process
4. Information Hiding
5. Cohesion and Coupling

SOFTWARE DESIGN METHODS

Systematic approaches to developing a software design.

Structured Methods

- *Process functions are identified*

Object-Oriented

- *Develop an object model of a system*

Data-Oriented

- *Entities are determined for each sub-system*
- *Then entity inter-relationships are examined*

Component-based

- *Divide the system into components*

Formal Methods

- *Requirements and programs are translated into mathematical notation*

WHICH METHOD TO CHOOSE?

Data oriented Design:

Useful for systems that process lots of data, e.g. database and banking applications

Structured Design

Useful for process intensive systems that will be programmed using a procedural language such as C.

OO methods

Useful for any system that will be programmed using an object oriented language such as C++.

Component-based Methods

Useful for the large systems that can be modularized

Formal methods

Their use is expensive and claims of reduced errors remain unproven

However, the ability to formally validate the correctness of a software artifact is appealing

SOFTWARE DESIGN STRATEGIES

- *Divide-and-conquer/stepwise refinement*
- *Top-down vs. bottom-up*
- *Data abstraction and information hiding*
- *Use of heuristics (trial & error)*
- *Use of patterns and pattern languages*
- *Iterative and incremental approach*

DESIGN DOCUMENTATION

- *“Design without documentation is not a design”*
- *The design is usually documented as a set of graphical models.*
- *Possible models*
  - *Data-flow model*
  - *Entity-relation model*
  - *Structural model*
  - *Object models*

MODULAR DESIGN - BENEFITS

- *Easier to manage*
- *Easier to understand*
- *Reduces complexity*
- *Delegation / division of work*
- *Fault isolation*
- *Independent development*
- *Reuse*

CONCEPTS IN SOFTWARE DESIGN PROCESS

- ***Abstraction***
  - *Concentrate on a problem at some level of generalization without regard to irrelevant low level details*
- ***Refinement***
  - *Top down design strategy that successively refines the levels of procedural details*
  - *Every refinement step involves Design Decisions*
- ***Modularity***
  - *Divide the software into separately named components, that are integrated to satisfy the problem requirements*

EFFECTIVE MODULAR DESIGN

- *How to decompose a software system into best set of modules?*
  - *Information hiding*
  - *Functional independence*
  - *Cohesion*
  - *Coupling*

INFORMATION HIDING

- *Modules to be characterized by design decisions that hide from all others*
- *Design the modules in such a way that information (data & procedures) contained in one module is inaccessible to other modules.*
- ***Benefits:***
  - *When modifications are required, it reduces the chances of propagating to other modules.*

COHESION

A measure of interconnection among components of a single modules

“A module should ideally do one thing.”

Each module performs a single task requiring little interaction with other modules.

High cohesion is good

- *Changes are likely to be local to a module*
- *Easier to understand a module in isolation*

COUPLING

- *A measure of interconnection among modules in a software structure*
- *Depends on the interface complexity between the modules*
- *Number of dependencies between modules*
- *High coupling causes problems*
    - *Change propagation- ripple effect*
    - *Difficulty in understanding*

COHESION AND COUPLING

*Each module should be **highly cohesive***

–Components of a module are closely related to one another

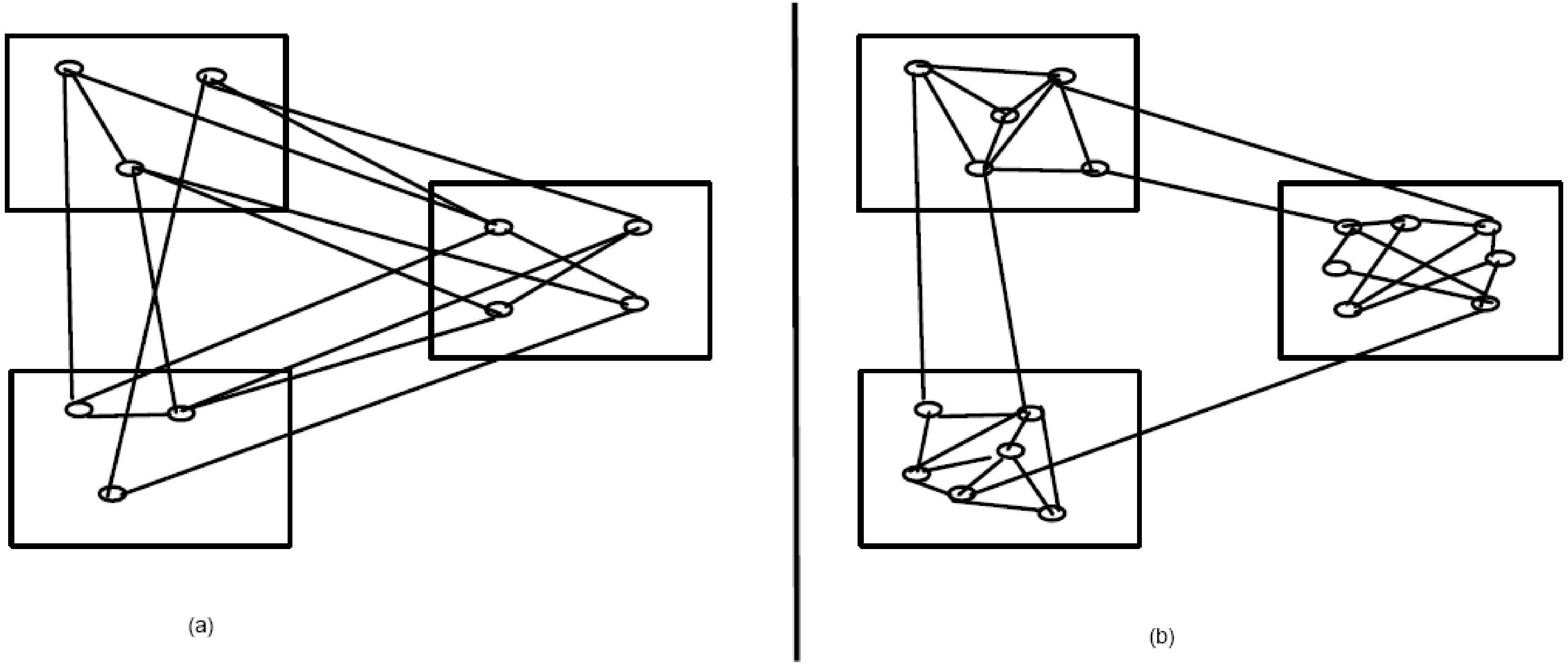
– Module understandable as a meaningful unit

*Modules should exhibit **low coupling***

–Modules should have low interactions with others

–Understandable separately

AN EXAMPLE



high coupling low cohesion

low coupling high cohesion

LECTURE CONTENTS

1. State Machine Diagram
2. Examples of State Machine Diagram
3. User Interface Navigation using State Machine Diagram

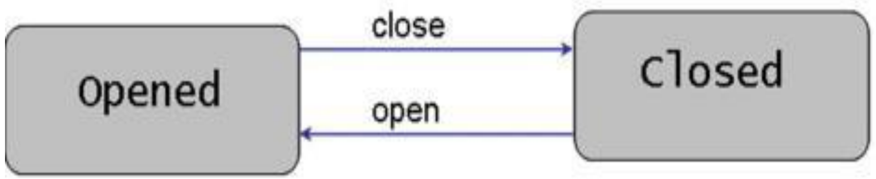
STATE MACHINE DIAGRAMS

- *As with activity diagrams, UML state diagrams show a dynamic view*
- *The UML includes notation to illustrate the events and states of things, transactions, use cases, people, and so forth*
- *It illustrates the interesting events and states of an object, and the behavior of an object in reaction to an event*

STATE MACHINE

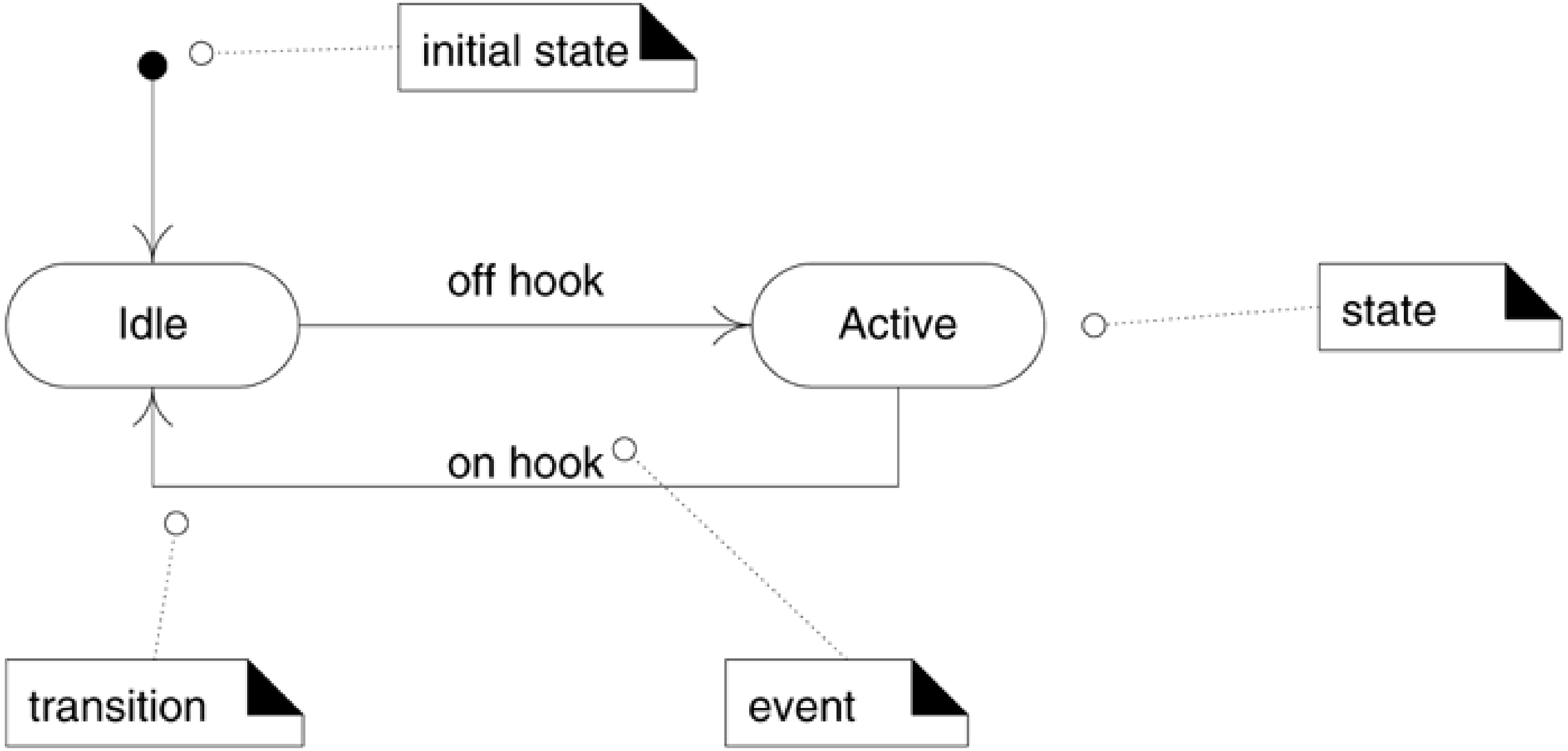
- State machine diagrams capture the behavior of a software system
- State machines can be used to model the behavior of a class, subsystem, or entire application.
- They also provide an excellent way to model communications with external entities via a protocol or event-based system.

EXAMPLE



- *The door can have either open state or closed during its life-cycle*
- *The door changes a state upon triggering of an <u>event</u>.*

STATE MACHINE DIAGRAM FOR A TELEPHONE



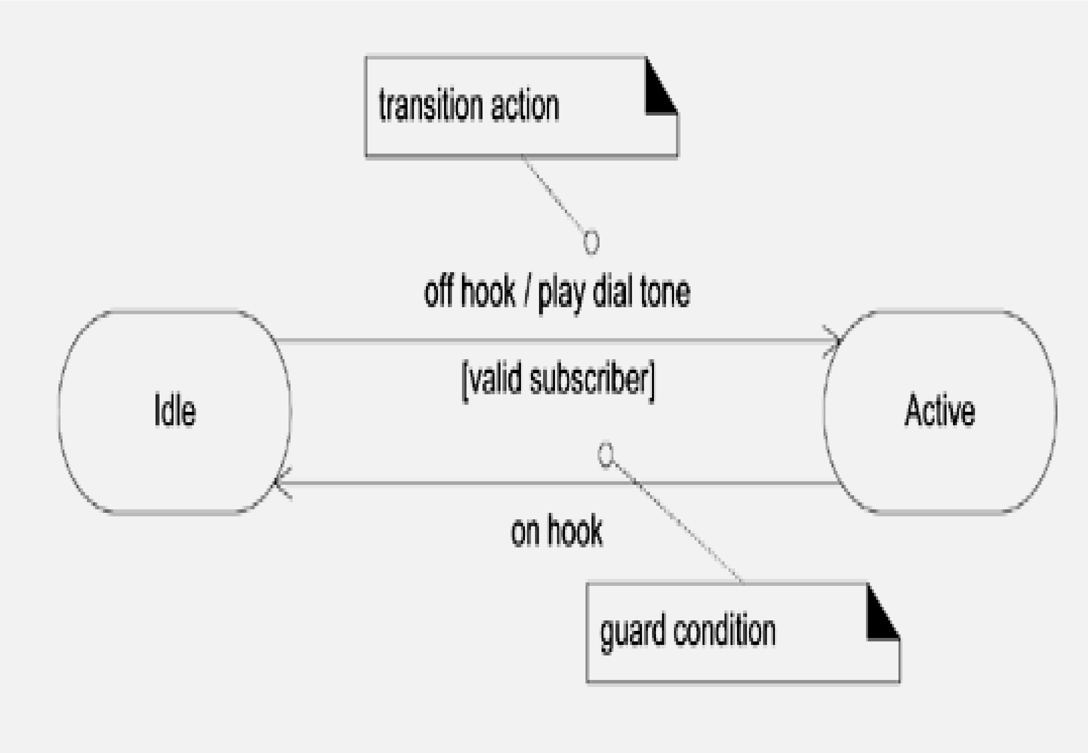
STATE MACHINE DIAGRAM

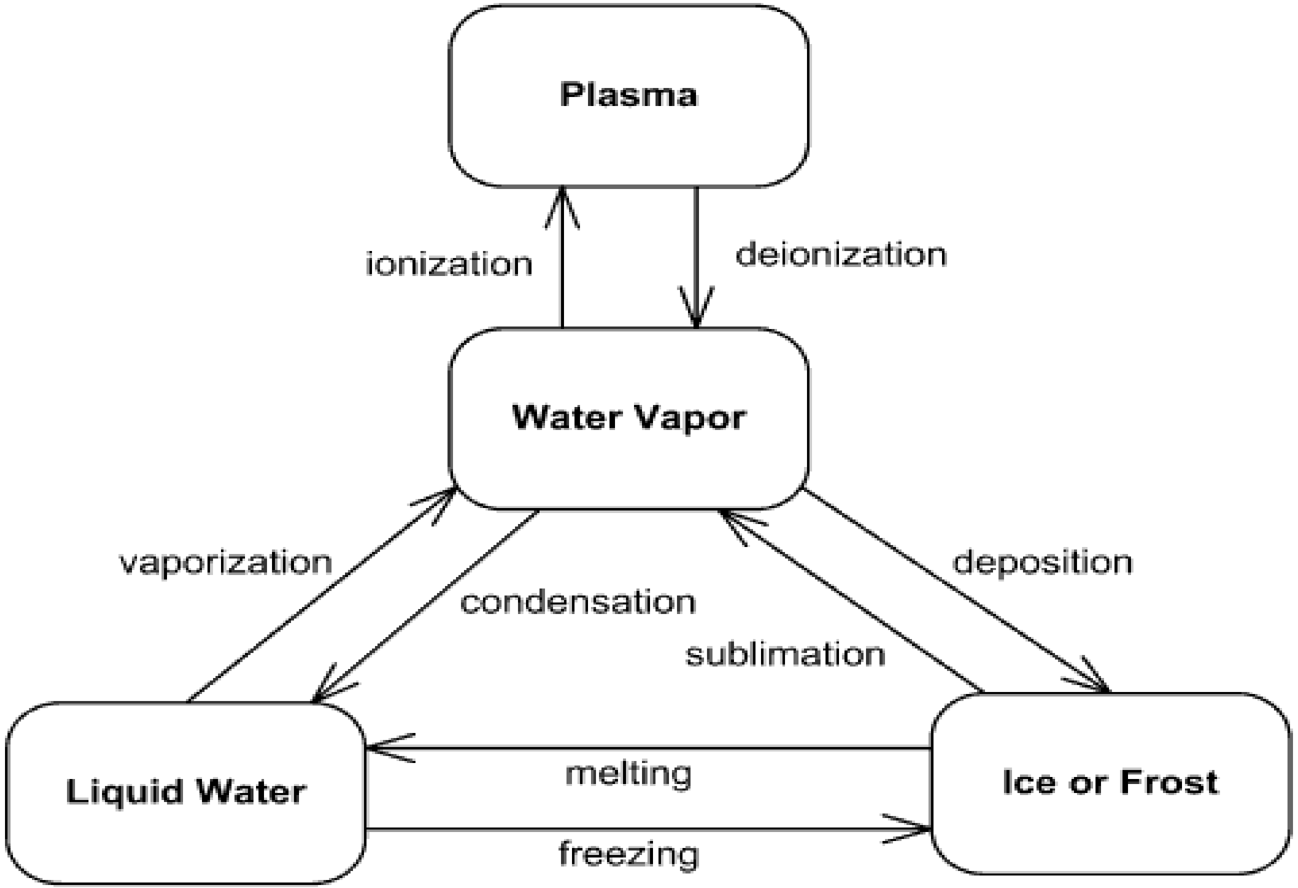
- *Transitions are shown as arrows, labeled with their event*
- *States are shown in rounded rectangles*
- *It is common to include an initial pseudo-state, which automatically transitions to another state when the instance is created*
- *A state machine diagram shows the lifecycle of an object:*
    - *what events it experiences*
    - *its transitions*
    - *the states it is in between these events*

STATE MACHINE DIAGRAM

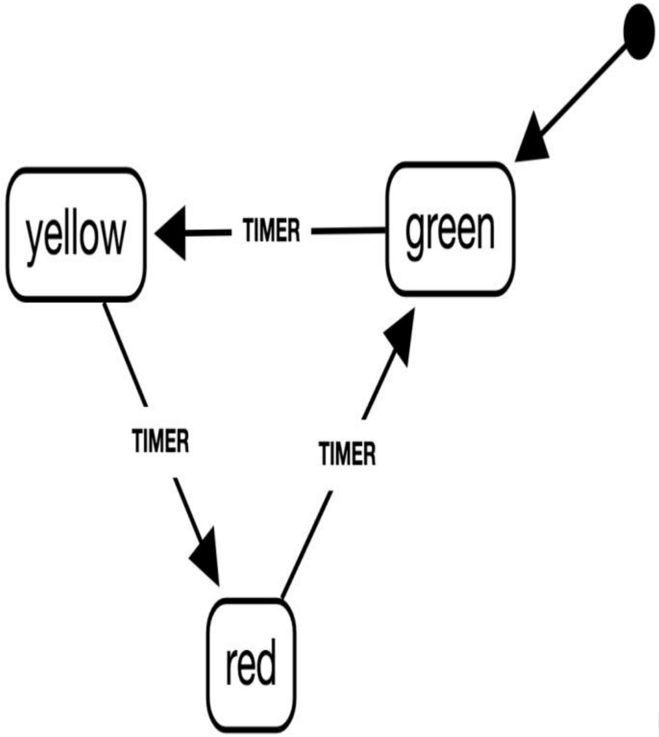
- *An **event** is a significant or noteworthy occurrence. For example:*
  - *A telephone receiver is taken off the hook.*
- *A **state** is the condition of an object at a moment in time between events. For example:*
  - *A telephone is in the state of being "idle" after the receiver is placed on the hook and until it is taken off the hook*
- *A **transition** is a relationship between two states that indicates that when an event occurs, the object moves from the prior state to the subsequent state. For example:*
  - *When the event "off hook" occurs, transition the telephone from the "idle" to "active" state.*

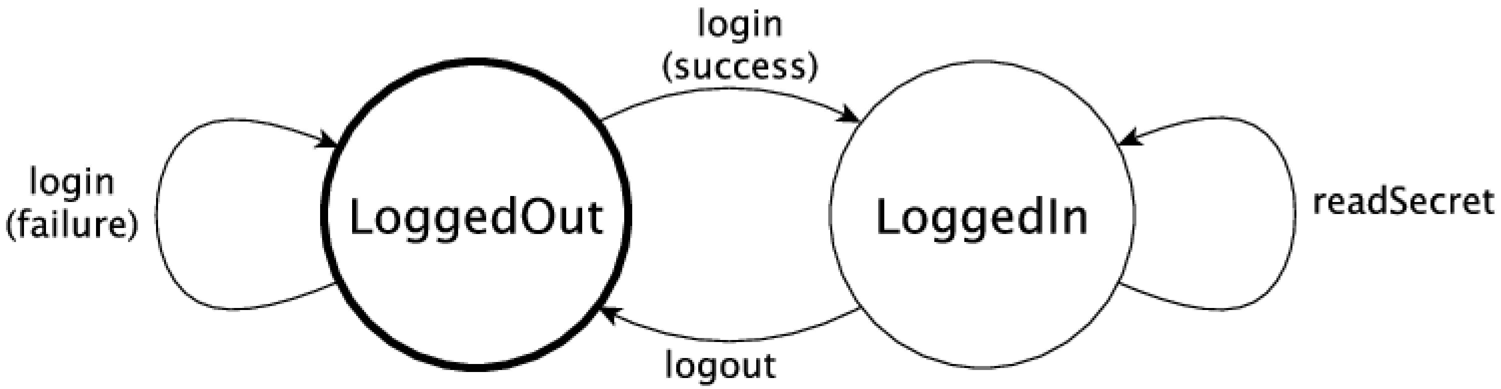
STATE MACHINE DIAGRAM

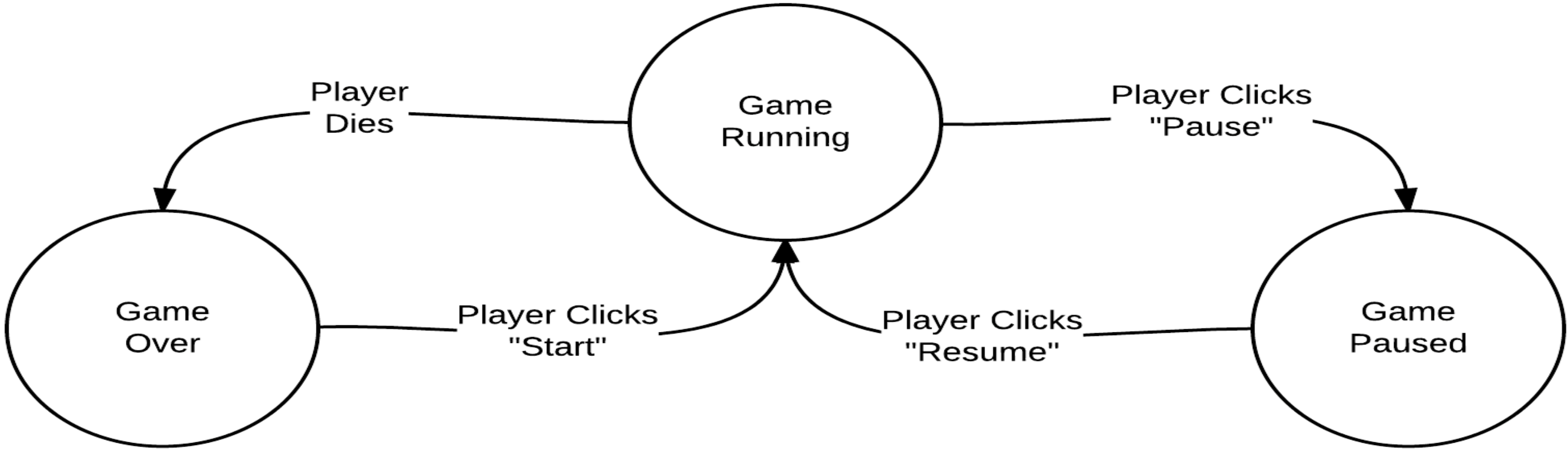




STATE MACHINE DIAGRAM


yellow
TIMER
green
TIMER
TIMER
red


login
(success)
login
(failure)
LoggedOut
LoggedIn
readSecret
logout


Game
Running
Player
Dies
Player Clicks
"Pause"
Game
Over
Player Clicks
"Start"
Player Clicks
"Resume"
Game
Paused