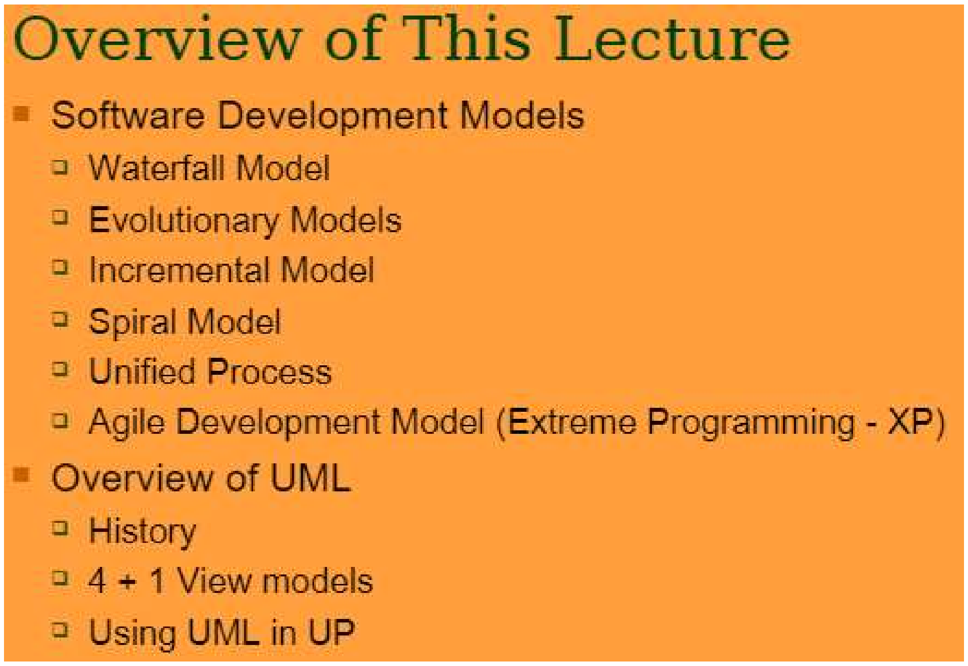
Lecture 2

Thursday, January 26, 2023 5:56 PM

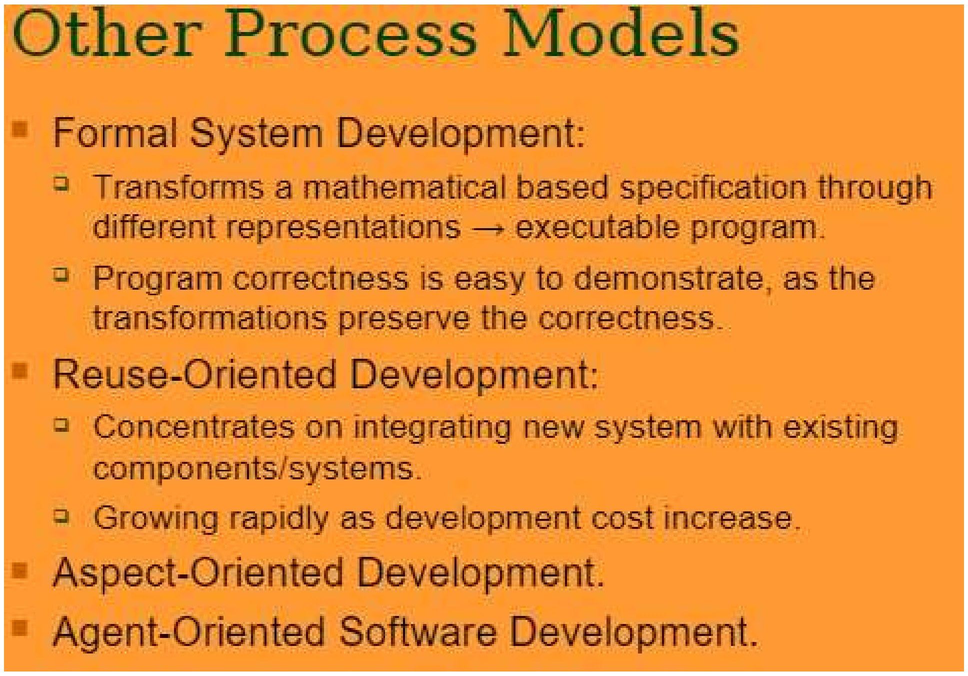
- 

Overview of This Lecture

- Software Development Models
  - Waterfall Model
  - Evolutionary Models
  - Incremental Model
  - Spiral Model
  - Unified Process
  - Agile Development Model (Extreme Programming - XP)
- Overview of UML
  - History
  - 4 + 1 View models
  - Using UML in UP

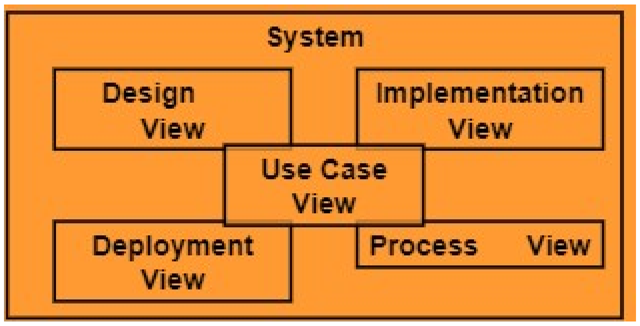
- Software Dev Models
  - High level software process is abstract rep of software dev, 4 activities
    1. Specification
    2. Development
    3. Validation
    4. Evolution
  - Theoretical framework usually extended & adapted from real world app, 2 generic models
    - Waterfall Model- earliest software dev model
      - Defined a number of phases, which go along w/ the 4 stages (^)
      - Assumption for model is phase takes place in sequence to another, each activity is completed b4 next starts
      - In theory, each phase makes docs that are verified, validated, and assumed to be complete
      - In theory, each phase depends on docs of prior stage to go forward (complete the stage b4)
      - In practice, stages overlap, feedback to each other
      - Advantages: tangible products (docs) @ end of every phase are good progress checker (plan b4 action emphasis, top-down perspective, big picture)
      - Disadvantages/Problems: specification frozen early (bc costly and time consuming, later stages can be carried out) , cannot adapt to changing/wrong specs (ignore/code around, can't meet user req), testing @ end of dev
      - Observations: stages can be iterative, flexibility in coping w/ changing specs, early & frequent validation of software sys, later models respond to later observations
    - Evolutionary Model- evolves an initial implementation w/ user feedback
      - 2 main types:
        - Exploratory Dev: explores req & delivers a final sys, start w/ something understood & evolve by adding features (proposed by customers)

            - Throwaway prototyping: understand the req & dev a better req def, experimenting w/ bad understood req, usually make UI w/ minor/no functionality
        - Advantages: customer in process, early/frequent testing, good for small-med sized sys
        - Disadvantages: intangible, unpredictable process, bad sys structure, sys not converge to final version
- Incremental Model- combo of advantages of waterfall + evo models
    - Each increment is mini-waterfall
    - Prioritize service given by sys, maps req to increment based on priority, freezes req for current increment, each increment release is a working sys
    - Advantages: early utilization, early increments serve as prototypes, lower risk of overall project failure, basic services implemented first, model includes lots of testing thru dev
    - Disadvantages: hard to map req to small increments, hard to define basic services shared by all later increments, variant is AGILE method
- Spiral Model- formalize the evo model, avoid management shortcomings
    - Spiral process, each loop = 1 iteration = 1 process phase, each loop move away from time and cost center (so they increase as move out), each loop passes thru 4 quadrants:
        - Objective setting
        - Risk assessment and reduction
        - Dev and validation
        - Planning
    - Risk driven, no prescribe fix process, flexible
- Unified Process- state-of-art process, learning from history of b4 software dev
    - Integrating 2 diff insights (definitive activities & deliverables like waterfall model, iterative & incremental processes), project split into many phases, each workflow places diff emphasis on activities depending on current iteration
    - 4 phases:
        - Inception: plan project, eval risks
        - Elaboration: understand prob, design sys architecture, plan dev
        - Construction: designing, pgrming & testing
        - Transition: moving sys from dev to user env, acceptance testing, release of full sys
- Agile Dev Model (XP)
    - Type of incremental model, software dev in incremental, rapid cycles, result in small releases w/ each building on previous functionality, each release is tested, time critical applications, extereme pgrming uses it
    - Advantages: customers happy, devs happy, communication btwn, software delivered consistently, lots of attention to detail
    - Disadvantages: difficult to assess effort, not much on designing and doc, easily taken off track if not good comm, senior pgrms can only make most of req decisions, no new pgrmrs allowed (unless combo exp w/ resources)

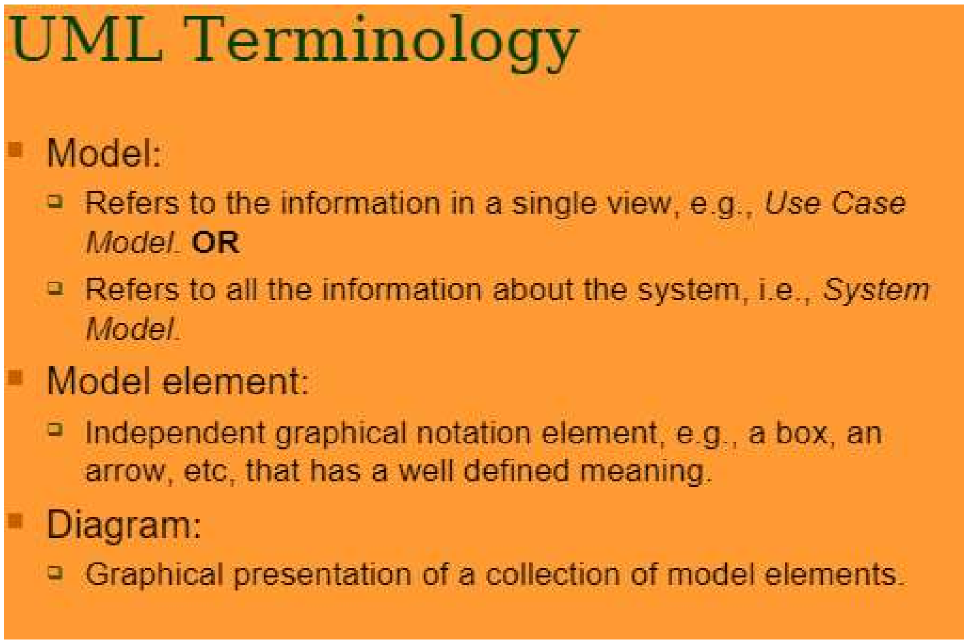
- ○ 

Other Process Models

- Formal System Development:
  - Transforms a mathematical based specification through different representations → executable program.
  - Program correctness is easy to demonstrate, as the transformations preserve the correctness.
- Reuse-Oriented Development:
  - Concentrates on integrating new system with existing components/systems.
  - Growing rapidly as development cost increase.
- Aspect-Oriented Development.
- Agent-Oriented Software Development.

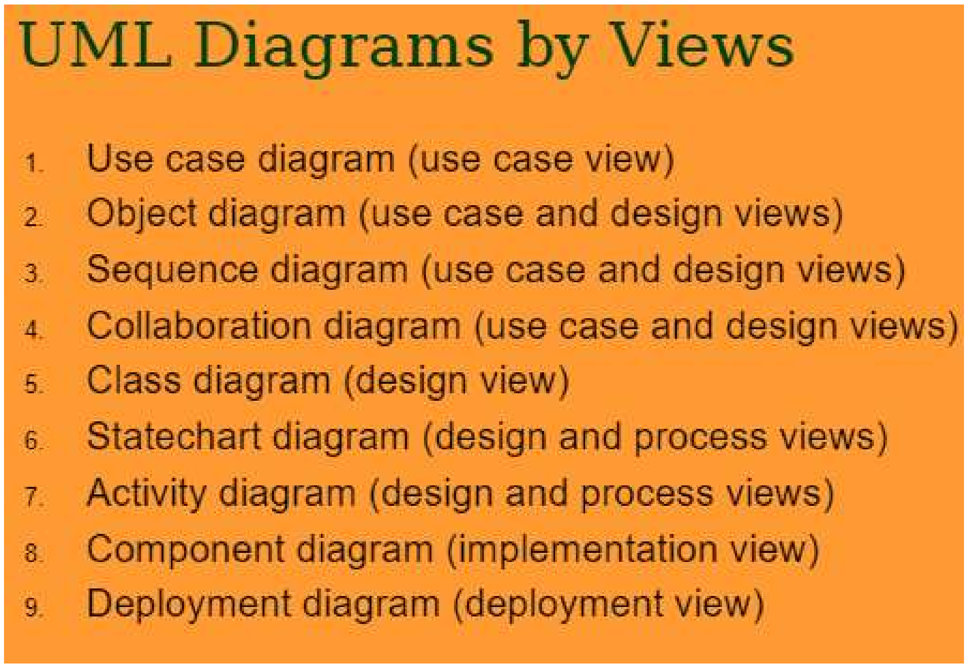
- Overview of UML (unified modeling lang)
  - Visual lang to visualize & make doc software sys
  - Has grammar and semantics, used w/ oop
  - Currently industry standard, sep lang from software process & can be used w/ other software models
  - Not a pgrming lang, or software modeling tool, not a SE method/ software dev process
  - 4 + 1 models-
  - 

  - Use case view- for sys analyst & end users & testers, used for defining req of sys, describing the sys external behavior, big role in driving dev process
  - Design View- for sys analyst & pgrmers, for describing support of functional req in case view, made of def of pgrm parts & behavior & interactions & useful as basis for coding
  - Implementation View- for sys engineer & tester, for describe physical parts that make up sys, useful for config management & sys integration
  - Process View- for sys analyst & pgrmer & tester, for non-functional req, define things in sys
  - Deployment View- for sys integrator (client side), nonfunctional, for describing physical parts that deployed in physical env



UML Terminology

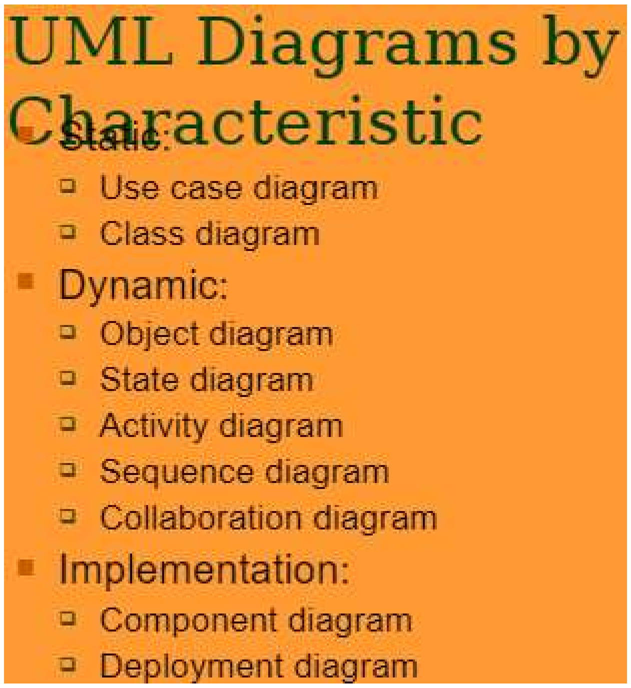
- Model:
  - Refers to the information in a single view, e.g., *Use Case Model*. **OR**
  - Refers to all the information about the system, i.e., *System Model*.
- Model element:
  - Independent graphical notation element, e.g., a box, an arrow, etc, that has a well defined meaning.
- Diagram:
  - Graphical presentation of a collection of model elements.



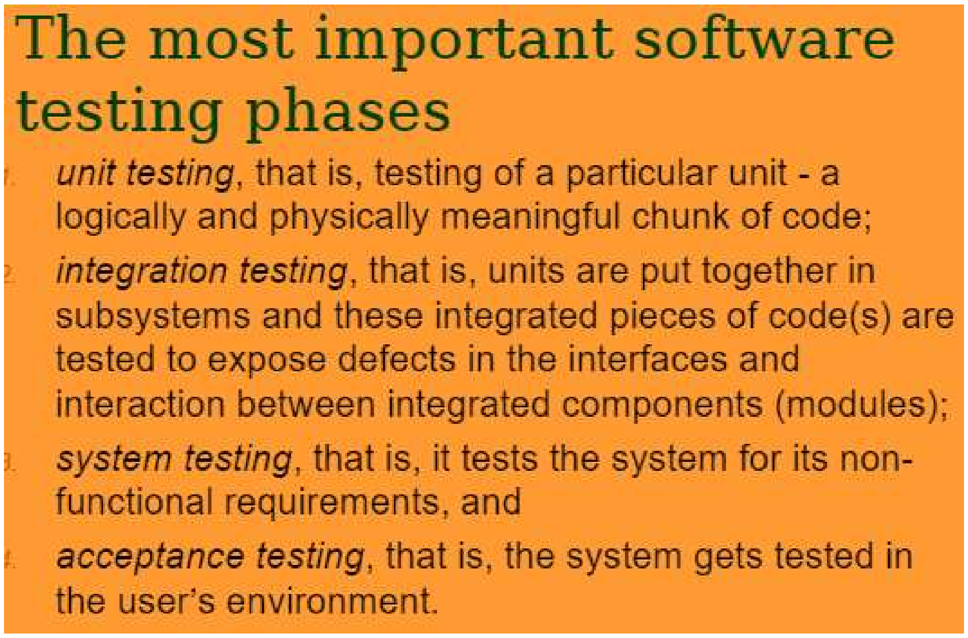
UML Diagrams by Views

1. Use case diagram (use case view)
2. Object diagram (use case and design views)
3. Sequence diagram (use case and design views)
4. Collaboration diagram (use case and design views)
5. Class diagram (design view)
6. Statechart diagram (design and process views)
7. Activity diagram (design and process views)
8. Component diagram (implementation view)
9. Deployment diagram (deployment view)

- By chars- software sys has 2 chars: static (logical) and dynamic (behavior of sys)

- 

  ## UML Diagrams by Characteristic

  - Static:
    - Use case diagram
    - Class diagram
  - Dynamic:
    - Object diagram
    - State diagram
    - Activity diagram
    - Sequence diagram
    - Collaboration diagram
  - Implementation:
    - Component diagram
    - Deployment diagram
- UP (unified process) & UML
  - UP is case driven
  - UML diagrams used in req, analysis (use cases describing how user interact w/ sys), & design (analysis & design overlap in UP, go on by realization & refinement) activities in UP workflow
    - Realization tell how functionality supported by sys, cause domain model to be refined into more implementation-oriented class diagram
-
- 

  ## The most important software testing phases

  1. *unit testing*, that is, testing of a particular unit - a logically and physically meaningful chunk of code;
  2. *integration testing*, that is, units are put together in subsystems and these integrated pieces of code(s) are tested to expose defects in the interfaces and interaction between integrated components (modules);
  3. *system testing*, that is, it tests the system for its non-functional requirements, and
  4. *acceptance testing*, that is, the system gets tested in the user's environment.
-

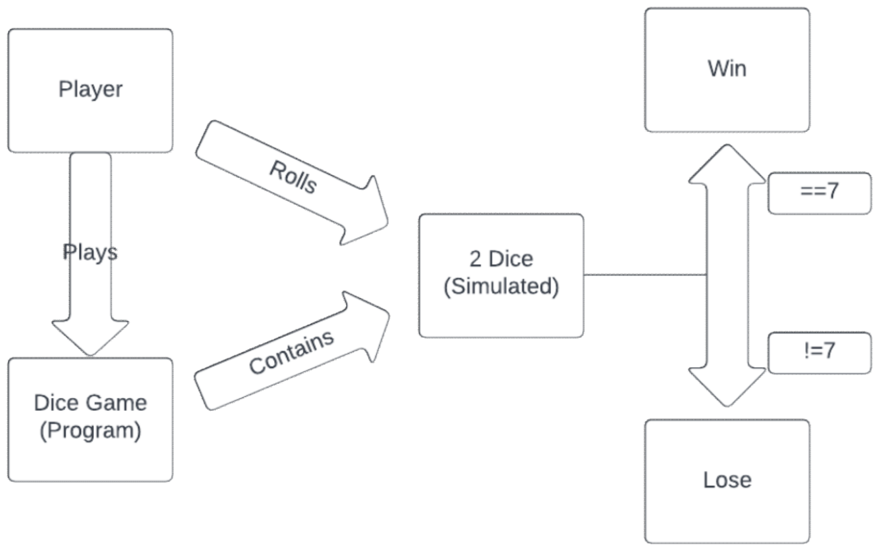
Lesson 2 Assessment

Thursday, January 26, 2023 8:07 PM

Question 1
10 Points

(page 8, [Larman, 2005]) Define the use cases and domain model for the “dice game”, in which software simulates a player rolling two dice. If the total is seven, then the player wins, otherwise he loses.

Use Case: A player uses software to role two simulated dice. If the dice combined equal seven the player has won, otherwise, he has lost.
Domain Model:



Question 2
10 Points

Discuss what are the software testing steps and why are these important.

- Unit testing- important to see if a 'chunk' of code works as it was designed to be
- Integration testing- important to ensure subsystems cooperate
- System testing- important to test for the system's non-functional requirements
- Acceptance testing- important to check the user's environment

Question 3
10 Points

Which software model is iterative and incremental in nature and emphasizes creation of models rather than paper-documents? Explain the phases of this process.

The incremental model Is incremental in nature and emphasizes the creation of models. The phases include increments that slowly progress. The first increment (or first few) satisfies the most crucial requirement. All following increments that follow are done in priority order. Each increment is a mini-waterfall as well to help with prioritizing.

Question 4
10 Points

Which software process model recognizes importance of risk management in a project? Explain.

The spiral model recognizes the importance of risk management as it sets aside time for risk assessment. It explicitly identifies risks for each iteration.

Question 5
10 Points

Which of the following is NOT a software process model?

The Waterfall model;
The Spiral model;
The Rainfall model;
The Unified Process model.