Access to Project 2

- Please send us an email with the repo link for the forked project

- Ensure we have access to your Project 2 repo

Project 2 Evaluations

- Three different evaluations:
  - Self evaluation (scale: 0 to 10) - individual submission
  - Team evaluation (scale: 0 to 10) - individual submission
  - Code Base Evaluation - **group submission**
    (4 criteria, scale: 0 to 5 for each criterion)
- Available through Blackboard (under *Assignments*)
- Deadline: **Friday, March 9, 11:59PM CDT**

Project 2 Presentations

- Presentation schedule is available on the course webpage
- Each presentation should last **at most 10 minutes (including setup).**
- Check-out:
  - Project 2 description for details on what the presentation should contain
  - Your Project 1 feedback (on Blackboard)
  - “Project 2 Overview / Advice for Project 2” slide set

https://www.ittc.ku.edu/~alexbardas/eecs448/materials/projects/project2-presentations.pdf

Lab5

- Available on the course webpage (under *Course Schedule*)

- Common deadline for all lab sections:

  March 16 at midnight (11:59pm CDT)

- No lab sessions next week (March 12 -16)

Midterm Exam Review

Prof. Alex Bardas

Midterm Exam

- Time: **March 12 (Monday), 8:00 am – 8:50 am**
- Location: LEEP2 2415 (our regular classroom)
- Closed-book, closed-notes
- No electronic devices are allowed!
- **One** “note” sheet allowed
  - Letter-size or A4 , double-sided: handwritten or typed

Format

- The exam has
  - Multiple choice questions (“one answer”)
  - True/False questions
  - Right sequence questions
  - Short answer questions (One phrase to at most a paragraph)
  - Diagram(s) questions:
    - Drawing a diagram based on a given description
    - Tasks/questions based on a given diagram

What will the Midterm Exam cover? (1/2)

- **Unit 1: Basic Software Engineering Concepts**
  - Slide set:
    - Introduction to Software Engineering
  - Textbook: Chapters 1 and 2
- **Unit 2: Software Processes**
  - Slide sets:
    - Software Processes (Overview, Process Models: Prescriptive vs. Agile, Prescriptive Models)
    - Agile Development (Process Models: Agile Models, Agile: XP, SCRUM, DSDM, AUP)
  - Textbook: Chapters 3, 4, and 5
  - Additional reading materials – available on the *Course Schedule* (course webpage)

What will the Midterm Exam cover? (2/2)

- **Unit 3: Requirement Modeling**
  - Slide sets:
    - Requirements Analysis
    - Requirements Modeling: Scenario-Based
    - Requirements Modeling: Domain Models
    - Requirements Modeling: Class-Based and UML Class Modeling
    - Requirements Modeling: Behavior-Based
  - Textbook: Chapters 8, 9, 10, and 11
  - Additional reading materials – available on the *Course Schedule* (course webpage)
- **Unit 4: Project Management Concepts**
  - Slide set:
    - Project Management Concepts
  - Textbook: Chapters 31

A Brief Overview of the Four Units

Unit 1: Basic Software Engineering Concepts

- What is software?
- What are the characteristics of software?
- Why do we need software engineering?
- What is important to software engineers?
- What is the generic process framework for software engineering?

Why Software Engineering?

- Problem specification 

final program

- But …
  - Where did the specification come from?
  - How do you know the specification corresponds to the user's needs?
  - How do you decide how to structure your program?
  - How do you know the program actually meets the specification?
  - How do you know your program will always work correctly?
  - What do you do if the users need changes?
  - How do you divide tasks up if you have more than a one-person team?

Software Processes

- A software process is a set of **activities**, **actions**, and **tasks** that lead to the production of a software product
  - *Activities* are related to achieving a broad objective
    - Communication with stakeholders
  - *Action* produces a major work product
    - Architectural design
  - *Task* accomplishes a small, well-defined objective
- The process is focused on adaptively choosing an appropriate set of work actions and tasks

SE Process Framework

Software Engineering (IEEE definition):

1. The application of a **systematic, disciplined, quantifiable approach** to the **development, operation, and maintenance** of software; that is, the application of engineering to software.
2. The study of approaches as in 1.

- SE process framework establishes the foundation for a complete SE process

Process Framework

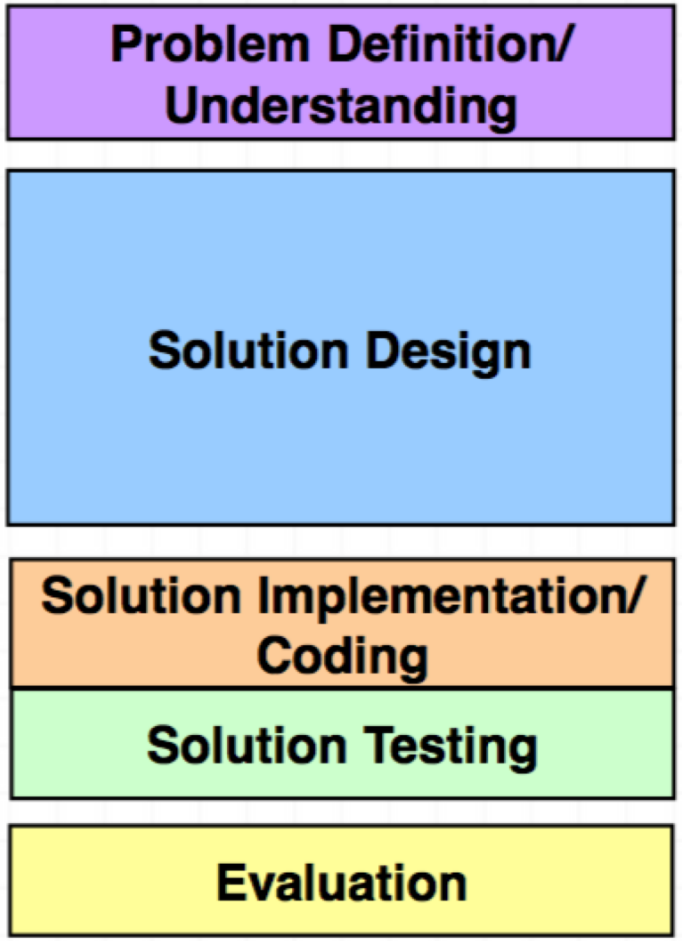
Framework activities:

- Work tasks
- Work products
- Milestones & deliverables
- Q&A checkpoints
- Deployment

Umbrella activities

SE Process Framework Activities

- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment
  - Delivery and customer evaluation



Umbrella Activities

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical review
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

Other Key Points

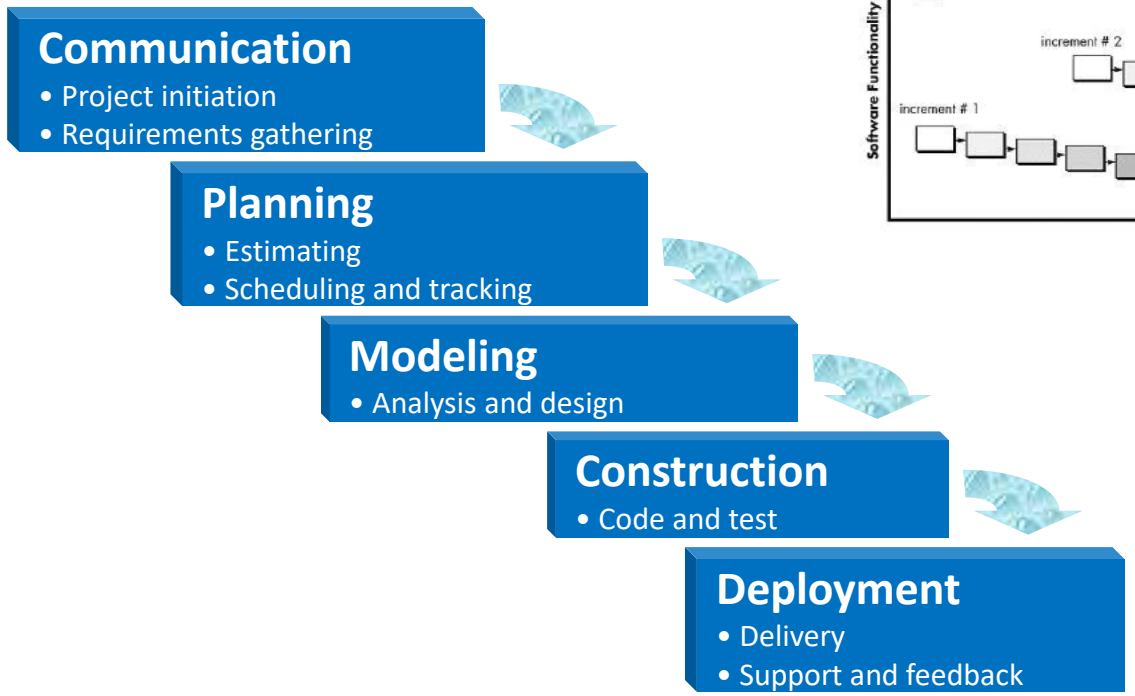
- Software is engineered not manufactured
- Software does not wear out, but it does deteriorate
- Software is complex and the development cost is high
- Problem should be understood before a software solution is developed
- Software should exhibit high quality
- Software should be maintainable

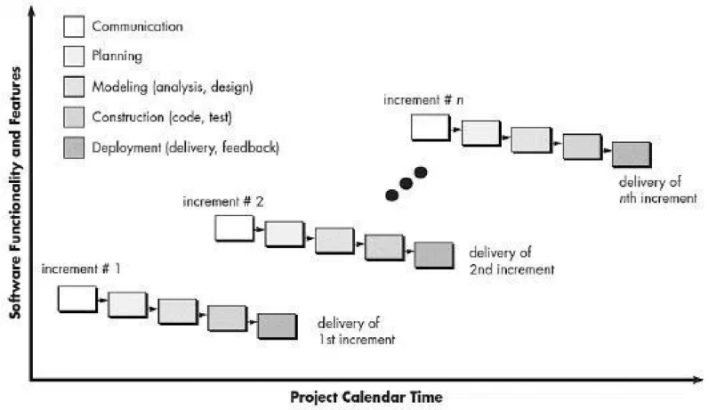
Unit 2: Software Processes

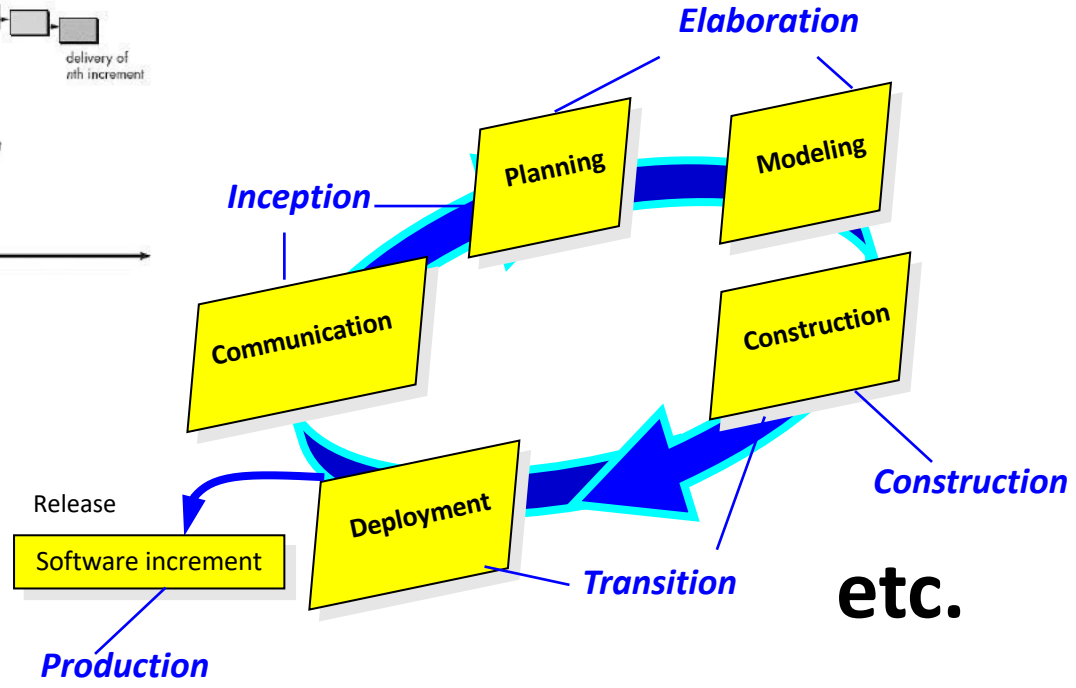
- What is a software process?
- Why do we need software process models?
- What are the characteristics of good process models?
- Four types of process flows
- What are the prescriptive models?
- What are the agile models?
- Prescriptive models vs. agile models

Process Models

- State the Pros and Cons for each process model we covered in class







For instance: Waterfall Model Pros & Cons

- Waterfall Model is the “old fashioned” software lifecycle model
- Pros:
  - It is best understood by upper management
  - It fits the cases where requirements are well understood and risk is low
  - It is often used with well-defined adaptation/enhancement of existing software
- Cons:
  - It doesn't support iteration: changes can cause confusion
  - It’s difficult for customers to state all requirements explicitly and up front
  - Customers need to be patient for a working version

Incremental Model Pros & Cons

Pros:

- Delivers small yet usable pieces, each is an increment on a previous piece
- Iterative in nature; multiple independent deliveries are identified
- Provides a needed set of functionality sooner while delivering optional components later
- Used when requirements are well understood
- Useful also when staffing is too short for a full-scale development

Cons:

- Work flow is still in a linear fashion within an increment
- It is staggered between increments

Evolutionary Model Pros & Cons

- What they are?
- Pros:
  - Evolutional delivery to adapt to frequent changes in the business and product requirements
  - Accommodate uncertainty better by delivering partial solutions in an orderly and planned manner
- Cons
  - Prototyping poses a problem to project planning because of the uncertain number of iterations required to construct the product
  - Evolutionary software processes do not establish the maximum speed of the evolution
  - Software processes should focus first on flexibility and extensibility than high quality

What is “Agility”?

- Effective (rapid and adaptive) response to **change**
- Effective **communication** among all stakeholders
- Drawing the **customer** onto the team
- Organizing a team so that it is in **control** of the work performed
- Keep the work product **essential** and **lean**

Yielding …

- **Rapid**, **frequent**, **incremental** delivery of software

Agile Process Models

For example, Extreme Programming (XP)

- **XP Values**
  - Communication (between team and with customers)
  - Simplicity (in design and code)
  - Feedback (at many levels)
  - Courage (to make and implement difficult decision, design for today)
  - Respect

XP Process (1/3)

The XP process is a 4-phase object-oriented approach

1. **XP Planning**
   - Begins with the creation of user stories
   - Agile team assesses each story and assigns a cost
   - Stories are grouped for a deliverable increment
   - A commitment is made on a delivery date
   - After the first increment "project velocity" is used to help define subsequent delivery dates for other increments

XP demands fixed time, not fixed features

XP Process (2/3)

2. XP Design

- Follows the KISS principle
- Encourages the use of Class-Responsibility-Collaborator cards
- For difficult design problems, suggests the creation of "spike solutions" – a design prototype
- Encourages "refactoring" – an iterative refinement of the internal program design

XP Process (3/3)

3. **XP Coding**
   - Recommends the construction of a unit test for a story before coding commences – test-driven development (TDD)
   - Encourages "pair programming"
     - Has pros and cons
4. **XP Testing**
   - Regression testing for unit tests
   - Integration and validation testing are executed daily
   - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality

Other Agile Process Models We Covered

- SCRUM
- Dynamic Systems Development Method (DSDM)
- AUP (Agile Unified Process)

Unit 3: Requirements Analysis

- What are the seven tasks in requirements engineering?
- What is a Requirements Traceability Matrix (RTM)?
- Types of requirements
- Requirements models
- What is the Domain Model?
- Structured vs. Object-oriented approaches
- UML diagrams:
  - use case diagram, activity diagram, swimlane diagram, class diagram, sequence diagram, and state diagram

Know their properties, similarities and differences

Requirements Engineering

- Accomplished through the execution of 7 major tasks
  1. **Inception** – roughly define scope
  2. **Elicitation** – define requirements
  3. **Elaboration** – further define requirements
  4. **Negotiation** – reconcile conflicts
  5. **Specification** – create analysis models
  6. **Validation** – ensure quality of requirements
  7. **Requirements management** – umbrella activities

Types of Requirements

- Functional requirements
  - Process and information: behavior, features, etc.
- Non-functional requirements
  - Operational: physical/technical environment
  - Performance: speed, capacity, reliability
  - Security: authorized access
  - Political and cultural factors affecting the system
- Design constraints
  - Choice of platform, programming language, etc.
- Process constraint
  - Resources, techniques, etc.

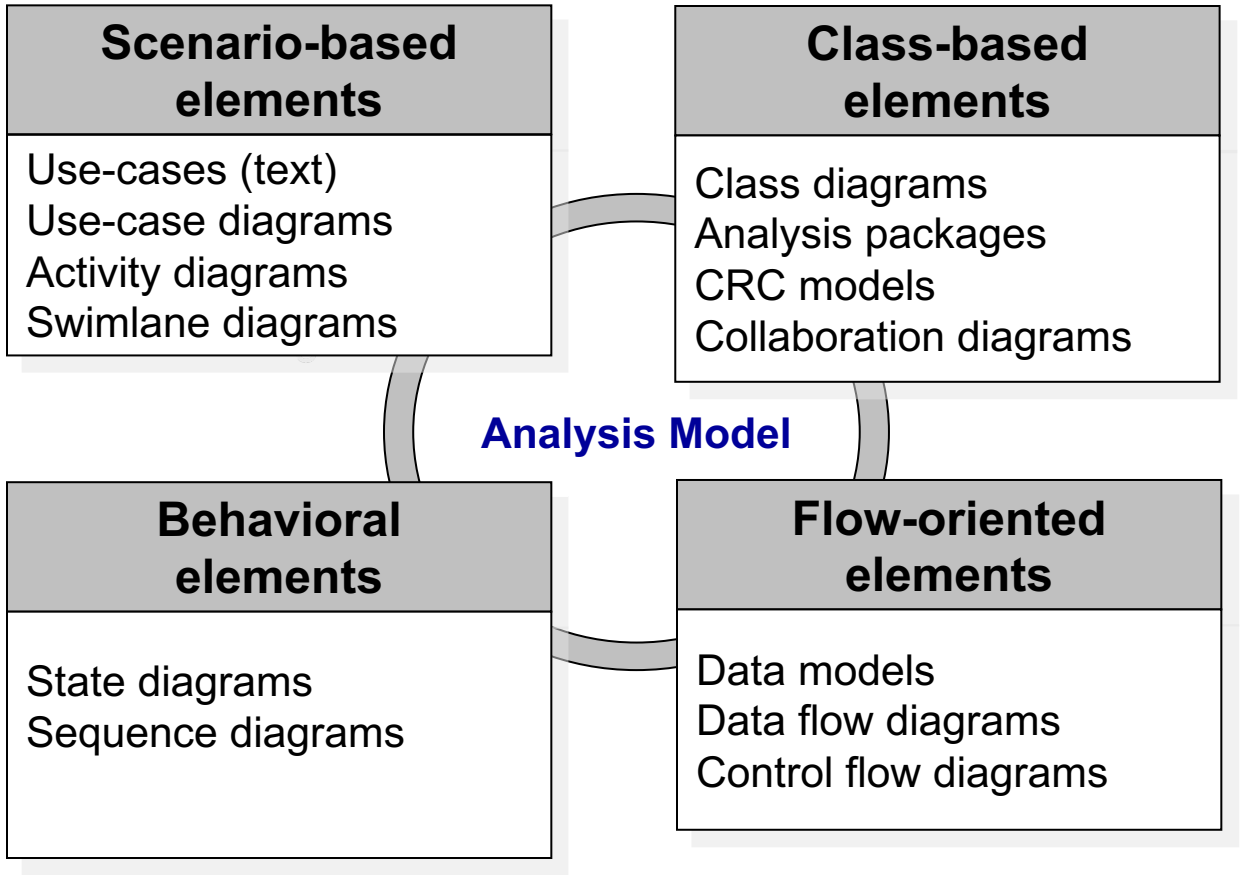
Traceability

- Traceability is the documented relationship between software engineering work products
- Requirements Traceability Matrix (RTM)
  - A set of tables that links requirements to system modules, and system modules to test cases
  - Rows: requirement names
  - Columns: work product names, test case names
  - An example: http://yaktrack.sourceforge.net/yaktrack_docs/a2332.html

Building the Analysis Model

- ***Requirements analysis*** elaborates on basic requirements to:
  - Specify software's *operational characteristics*
  - Indicate software's *interface* with other system elements
  - Establish the *constraints* software must meet
- It allows software engineers to build ***requirements models*** that depict
  - Scenarios viewed by actors → scenario-based models
  - Functional activities
  - Problem classes and their relationships → class-oriented models
  - System behavior trigged by events → behavior models
  - Information domain of the problem → data models
  - The flow of data as it is transformed → flow-oriented models
  - Constraints that software must meet

Elements of the Requirements Model



Scenario-Based Modeling

- **Use-cases** are simply an aid to defining what exists outside the system (actors) and what should be performed by the system.
  - It is a “contract for behavior”
  - How an actor uses the system-to-be to accomplish business goals
- **Steps:**
  1. List the *functions* performed by a specific actor
  2. Develop *primary scenarios* for each of the functions
  3. Evaluate for alternative behavior and develop a set of *secondary scenarios*
- Detailed use cases are usually written as *usage scenarios* or *scripts*, listing a specific sequence of actions and interactions between the actors and the system.

Domain Model

- A **domain model** is a conceptual framework of elements in the problem space
  - Uncovers entities and their static relations that make the black box behave as described by use cases
  - Starts from the “periphery” (or boundary) of the system
    - A boundary object translates information from an actor into a form that can be used by “internal” objects

Identifying Concepts

- Identify conceptual classes from noun phrases
  - Linguistic analysis: vision and scope, glossary, use cases
  - However,
    - Words may be ambiguous or synonymous
    - Noun phrases may also be attributes or parameters rather than classes

  e.g.,
    - If it stores state information or it has multiple behaviors, then it's a class
    - If it's just a number or a string, then it's probably an attribute
- Responsibilities can also be studied

Class-Based Modeling

- **Class-based modeling** represents:
  - ***Objects*** that the system will manipulate
  - ***Operations*** (a.k.a methods or services) that will be applied to the objects to effect the manipulation
  - ***Relationships*** between the objects
  - ***Collaborations*** that occur between the classes that are defined
- The elements of a class-based model include *classes* and *objects*, *attributes*, *operations*, *CRC models*, *collaboration diagrams* and *packages*.

Key Points

- What is an object/class?
- How to represent an object/class?
- What are the relationships between classes?
  - Association, generalization, dependency, aggregation, composition
- CRC modeling

Classes

- Entity classes
  - Extracted directly from the statement of the problem
  - Represent things to be stored or persist throughout the development
- Boundary classes
  - Create/display interface
  - Mange how to represent entity objects to users
- Controller classes
  - Create/update entity objects
  - Initiate boundary objects
  - Control communications
  - Validate data exchanged

Identifying Classes

6 selection characteristics

1. Retained information

   The potential class will be useful during analysis ***only if information about it must be remembered*** so that the system can function.

2. Needed services

   The potential class ***must have a set of identifiable operations*** that can change the value of its attributes in some way.

Identifying Classes

6 selection characteristics

3. Multiple attributes

During requirement analysis, the focus should be on “major” information.

A class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity.

4. Common attributes

A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.

Identifying Classes

6 selection characteristics

5. Common operations

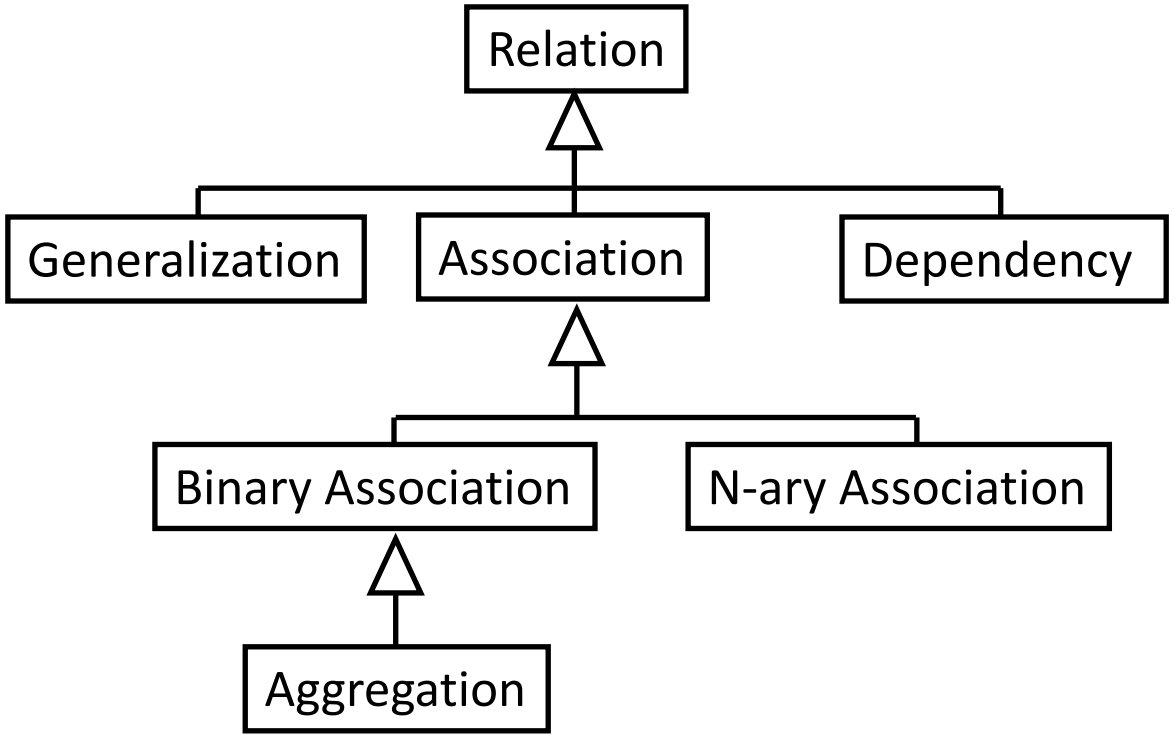
A set of operations can be defined for the potential class and these operations apply to all instances of the class.

6. Essential requirements

External entities that appear in the problem space and ***produce or consume information essential to the operation*** of any solution for the system will almost always be defined as classes in the requirements model.

Type of Relationships in Class Diagrams

- Class diagrams show relationships between classes.



Validating a Class Diagram

- One of the most important, and often overlooked issues is how to validate a class diagram.

- Given a specification or a use-case, can you look at the class diagram and use features of it to manually “execute” the use case?

UML Class Modeling

- An object-oriented modeling language developed in 1997
  - Models structure (static) and behavioral (dynamic) aspects of a system
  - Semi-formal: UML 2.0 added much more formality
  - Process-independent: can be used with a variety software development process models
  - Customizable and extensible

Abstraction Levels

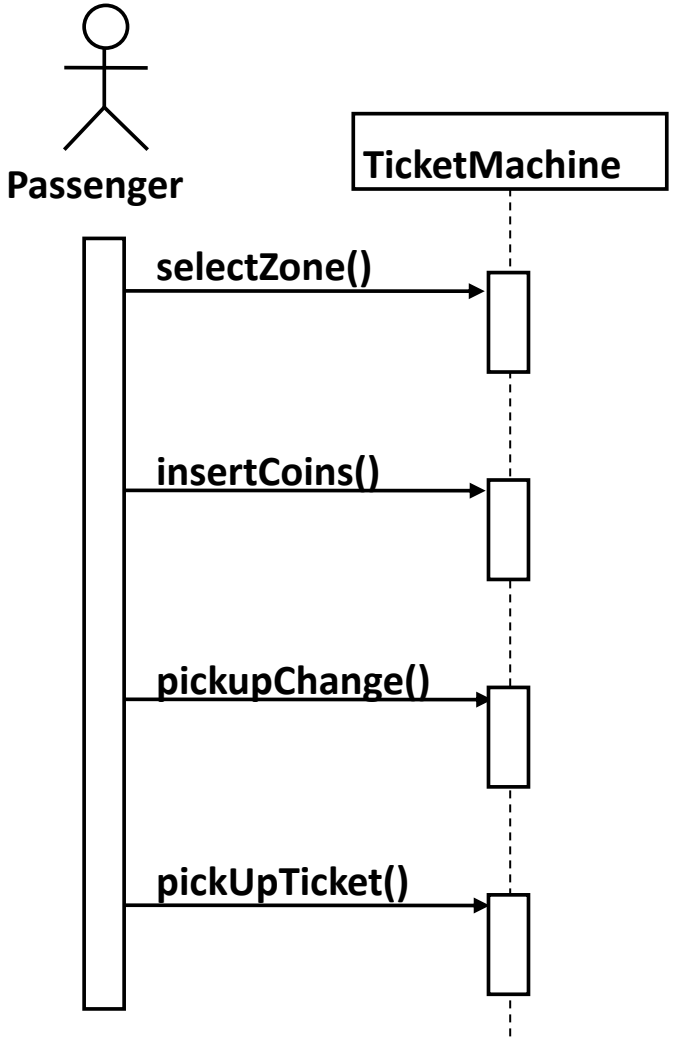
- Three perspectives for class models
  - Analysis
    - Represents concepts in the domain
    - Drawn with no regard for implementation (language independent)
  - Specification
    - Focus on interfaces not on how implementation is broken into classes
  - Implementation
    - A blue-print for coding
    - Direct code implementation of each class in the diagram

Behavioral Modeling

- Models the dynamics of the system
- Represents the behavior of the system as ***a function of events and time***
- Indicates how software will respond to events
  - Information being exchanged is not the essential part of the behavioral model but the fact that information has been exchanged

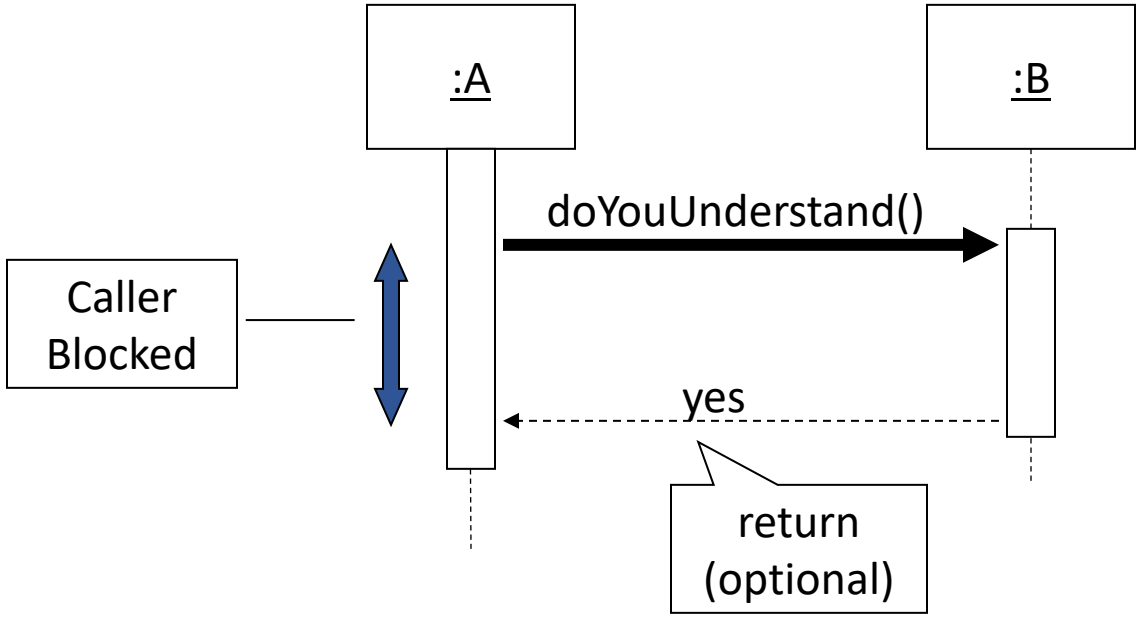
Sequence Diagram

- Objects: columns
- Messages: arrows
- Activations: narrow rectangles
- Lifelines: dashed lines



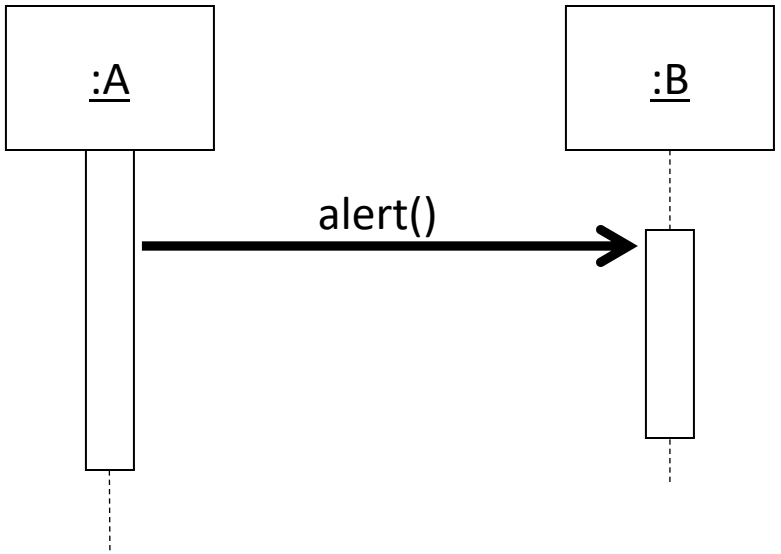
Sequence Diagram

- Synchronous message
  - The routine that handles the message is completed before the caller resumes execution



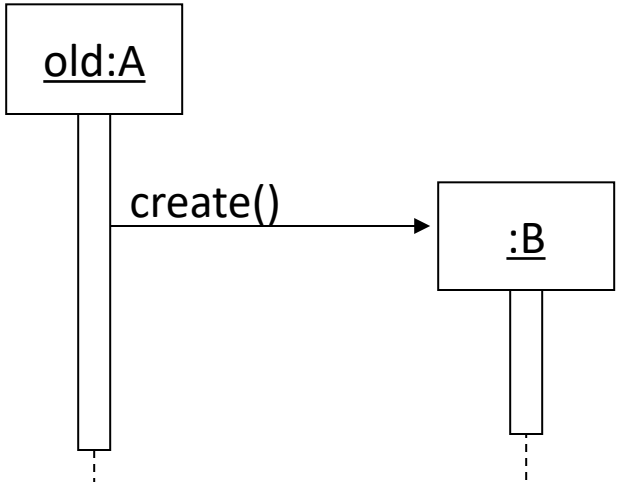
Sequence Diagram

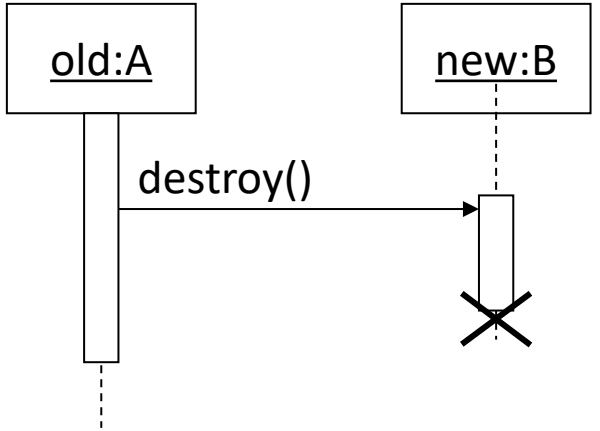
- Asynchronous message
  - Sender does not wait for the receiver to finish processing the message
  - Continues immediately



Sequence Diagram

- Message creation
  - Denoted by a message arrow pointing to the object
- Message destruction
  - Denoted by an X mark at the end of the destruction activation





Sequence Diagram

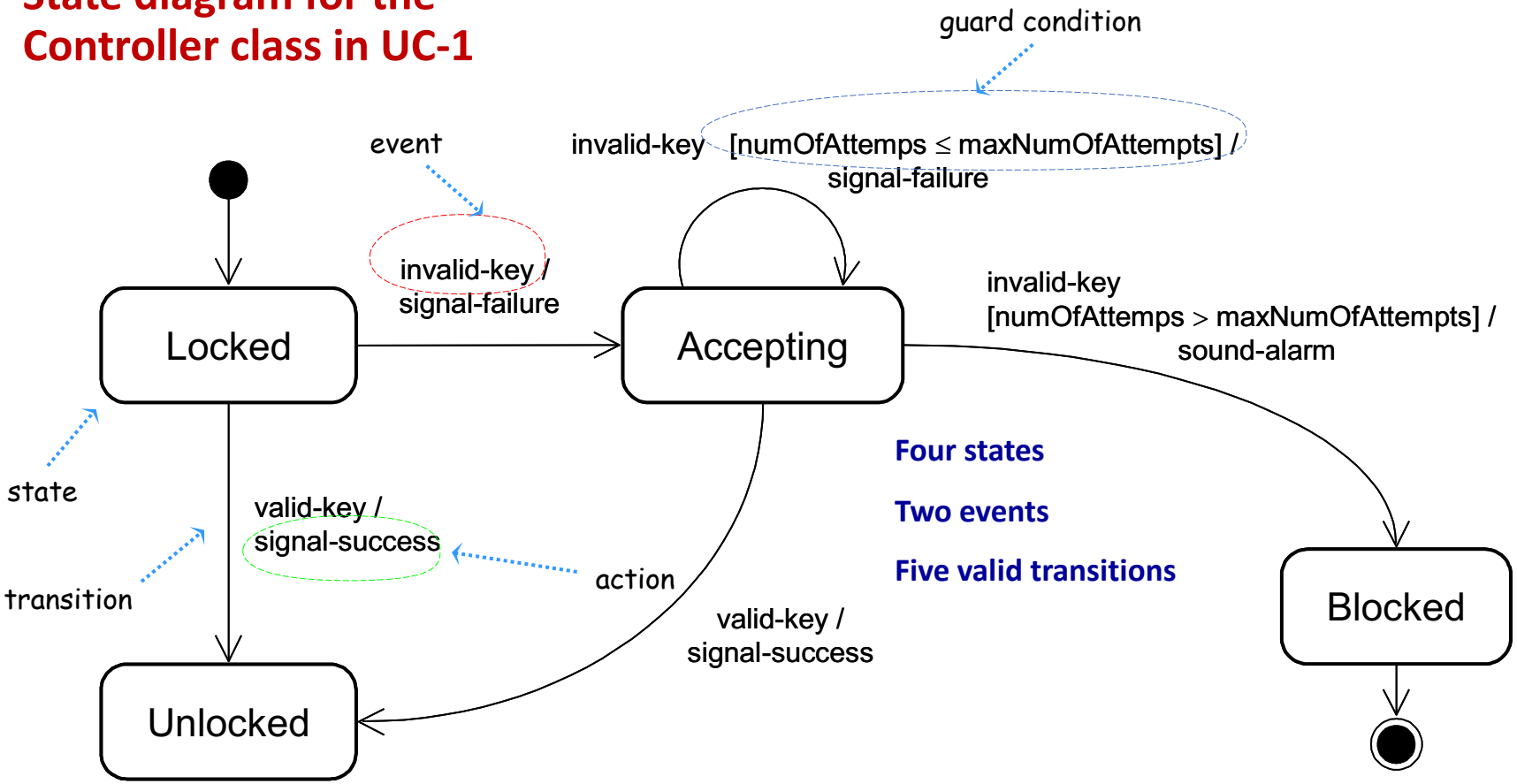
- Iteration
  - denoted by a * preceding the message name
- Condition
  - denoted by Boolean expression in [ ] before the message name

State Diagram

- Indicates *how an individual class changes state based on external events*.
- In other words, shows how a system or an object's behavior changes over time depending on the input
  - Shows the behavior of an object across several use cases
  - **One state diagram per class** to describe possible behavior for each instance of the class

A State Diagram Example

State diagram for the Controller class in UC-1



Unit 4: Project Management Concepts

- What are the four P's?
- What is MOI?
- What are Constantine's organizational paradigms?
- What are agile teams?
- What is the software scope?
- What is the common sense (commonsense) approach?
- What is the W5HH principle?

Start by checking out the "Project Management Concepts" slide set

Summary

- Structure and organize your note sheet (quality over quantity)
- **Read the questions and directions carefully**, answer what is asked
- Make a plan on how to manage your time during the exam

  e.g., First pass – spend less than a minute per multiple choice or true/false question