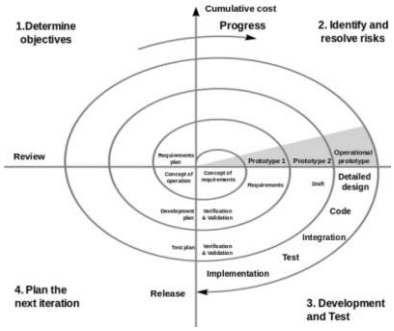
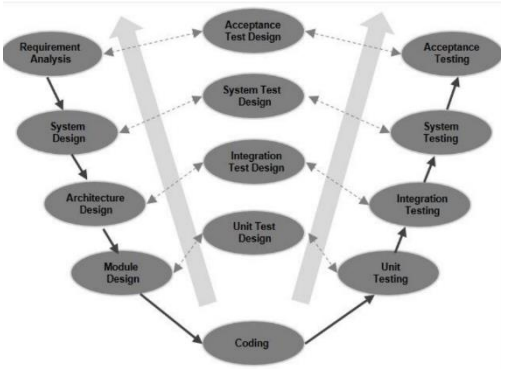
ENSF 613 – Summary Sheet

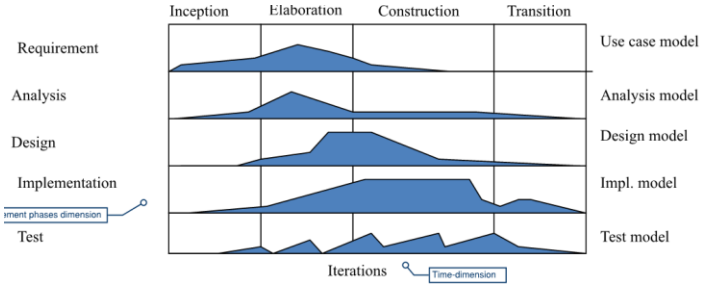
Software Development Process



- Software Development Life Cycle (DLC) includes inception (business rationale), knowledge elicitation, requirement analysis (who are users and what are inputs), design (architecture), construction, testing, maintenance.
- Waterfall Method is used when requirements are not expected to change, a system needs extensive documentation, or a project has a limited budget. Not suitable for complex projects.
- The Spiral Method phases include determine objectives, identify and resolve risks, development and test, and plan next iteration. Helpful for high risk projects where requirements are not clear, but expensive and complex. Addresses shortcomings of waterfall.
- Prototyping helps with communication and improves requirements and quality but may increase complexity and costs of system. Can be used stand-along or added into another process model.



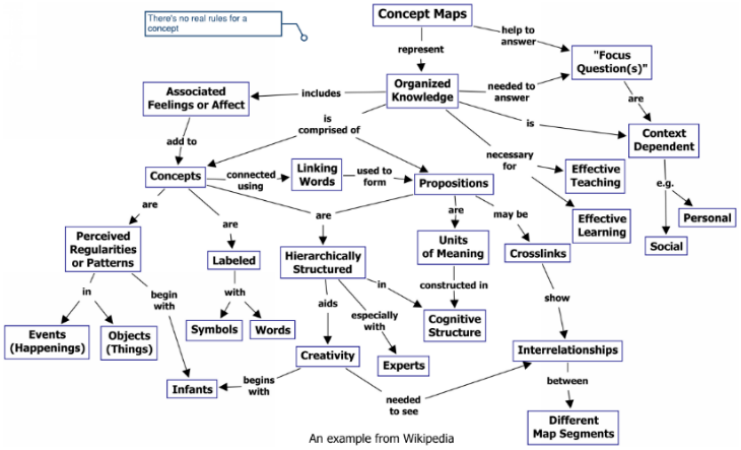
- V-Model focuses on validation and verification, and each phase must be complete before moving to next. Has a high success rate due to testing improving the quality, but it is not suitable for large projects since it is too rigid and risky.
- Test-Driven Development results in high-quality code where defects are caught early, but it is expensive to apply.
- The Rational Unified Development Process (RUP) is structures along two dimensions: Process Dimension and Time Dimension.
    - The inception phase studies the project's feasibility.
    - Elaboration phase collects more details requirements and creates the use case model and requirements.
    - Construction is the coding.
    - Transition is testing, training, and deployment.



- Agile approach creates a backlog of requirements through iteration, which are continuously prioritized and worked on during a time-box (Scrum) or sprint. It is assumed that requirements are likely to change using this method, and customers are actively involved with reviews.
- Extreme programming (XP), sometimes part of agile, includes TDD, simplicity, respect, and pair programming.
- Rapid Application Development (RAD) is like agile but relies on development tools to build prototypes for users and clients, who meet more frequently.

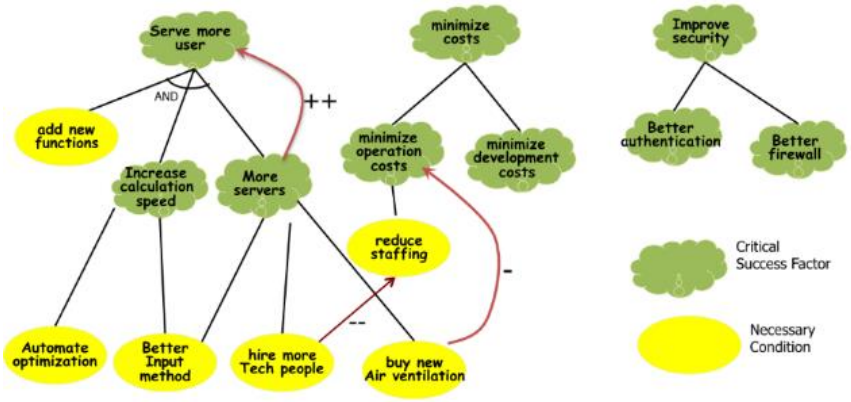
Knowledge Elicitation

- Functional requirements include what the product does, while non-functional are how the product can be described (light, usable, secure, etc.)
- Knowledge elicitation techniques include interviews, questionnaires, brainstorming, workshops, the use of diagrams (concept map, concept diagram, data flow diagram, use case diagram, state transition diagram, activity diagram, etc.)
    - A Concept Map doesn't really have rules, but just has a bunch of disparate requirements.



An example from Wikipedia

- Goal Trees look at relationships in between goals (helps +, hurts -, makes ++, or breaks --).



System Analysis

- The Net Present Value (NPV) calculation determines future earnings in today's dollars. The discount rate considers both the expected rate of return (say 12% per year), interest on debts, and inflation.
    - To get the NPV for each year, divide the net value (cash in + expected company value - cash out) for the year by (1 + discount rate)^n. Next, sum up the yearly NPVs, subtract the total by the initial present value, and divide by the initial present value.

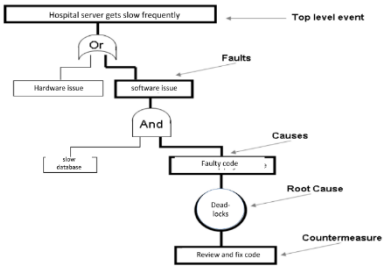
ROI = (FVI – IVI)/IVI

To calculate the present values you can use the follwing equation:

$$NPV = \sum_{t=1}^{n} \frac{Net\ Cash\ Flow}{(1+i)^t}$$

Where t is time (number of year), and i is the discount rate.

- Fault Tree Analysis uses Boolean logic to understand how a system can fail, starting with the most serious outcome at the top and working downwards using and/or gates to identify a root cause.
    - When calculating probabilities for one of the events occuring, you would multiply 'and' probabilities on the same level together, and add 'or' probabilities.

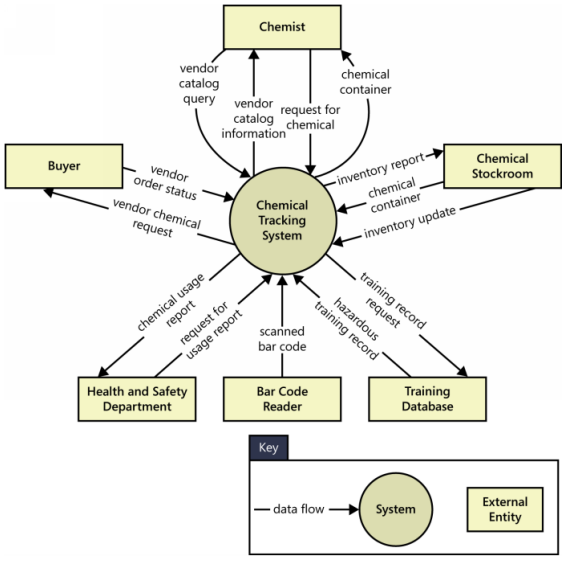


- Risk Analysis Matrices lay out the severity vs likelihood of risk events.

		Likelihood of Occurrence		
		Very likely	Possible	Unlikely
Sevirity	(4) Catastrophic			Flood
	(3) Critical		Power outage	
	(2) Marginal	Application Error		
	(1) Negligible	Slow server		

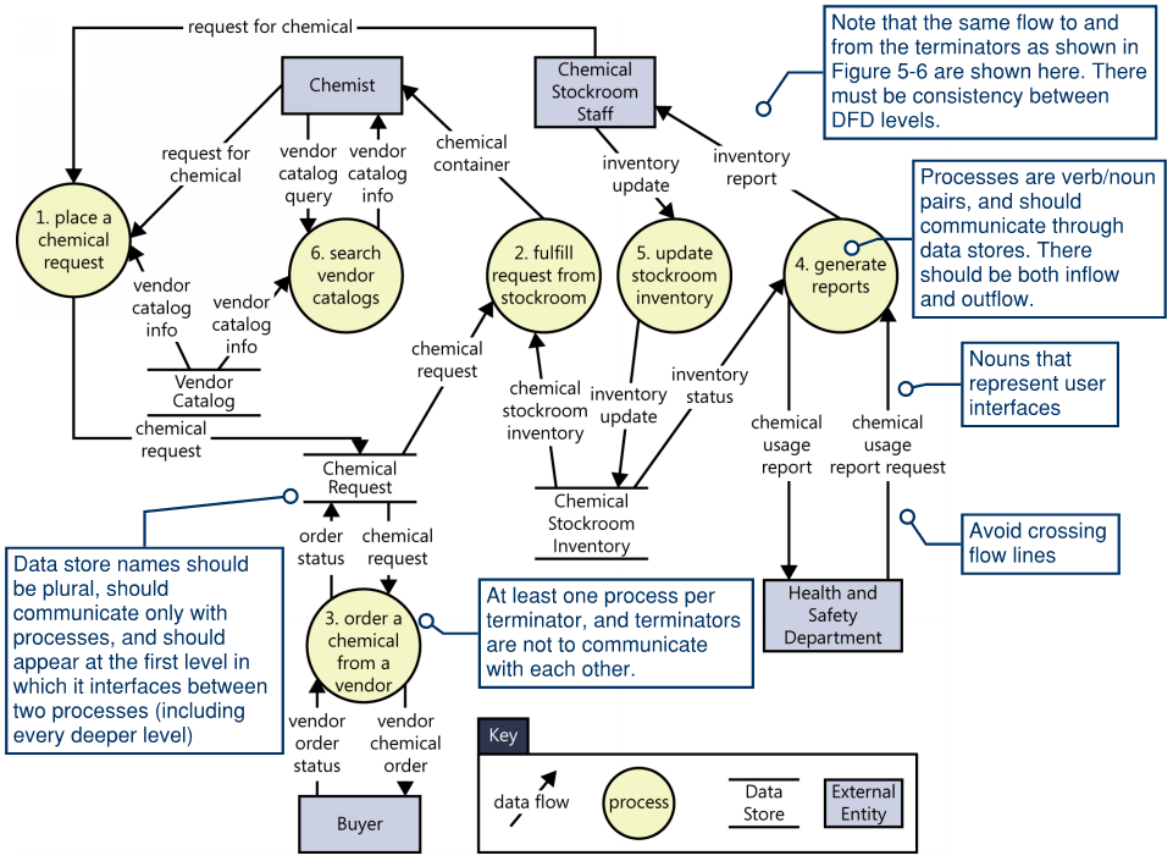
Structured System Analysis

- A Structured System Analysis is the analyzing and converting of business requirements into specifications, and then applications. Includes data flow diagrams, data models, and structured charts.
- A Context Diagram is the highest-level data-flow diagram.
    - Data flows (nouns) are shown between terminators (nouns) and the system (noun).
    - The context diagram identifies external entities (terminators) outside the system that interface to it in some way, as well as data, control, and material that flows between the terminators and the system.
    - Event types can be classified as flow, temporal, or control.
- A Data Flow Diagram (or Process Diagram) includes processes (verb-noun pairs), data flows (nouns), terminators (nouns), and data stores (plural nouns). There should be balance between levels of data flow diagrams.
    - We don't show how things are going to happen, only what things happen.
    - Data flows are typically flow, temporal (time-based), or control (signals).



    - Flows are often represented by complex data structures, which are defined in the data dictionary.
    - The details of how the data is transformed are better shown by steps in a process using state transition or swim lane diagrams.

- All data flows (arrows) from the context diagram also appear on the level 1 data flow diagram.
- Data stores should never be black holes or springs, so there should always be a data flow in and the exact same data flow out.
    - The data should not be changed (according to Dr. Moussavi), as only processes can change the data.

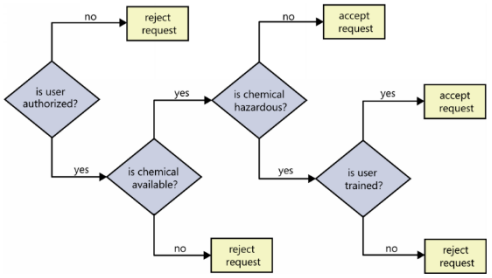


Process Specifications (PSPEC)

- Process Specs are sometimes written for low-level processes (2nd or 3rd level), and can be accompanied by Structured English, Decision Trees, State Transition Diagrams, etc.
    - A pre-condition for a process spec is what is required to invoke the process, while a post-condition is what it does.
- Structured English uses If-ElseIf-Else wording to break down a process into various outcomes.
    - Can use terms like BEGIN, IF, ELSE, ENDIF, DO WHILE, END DO, END/EXIT

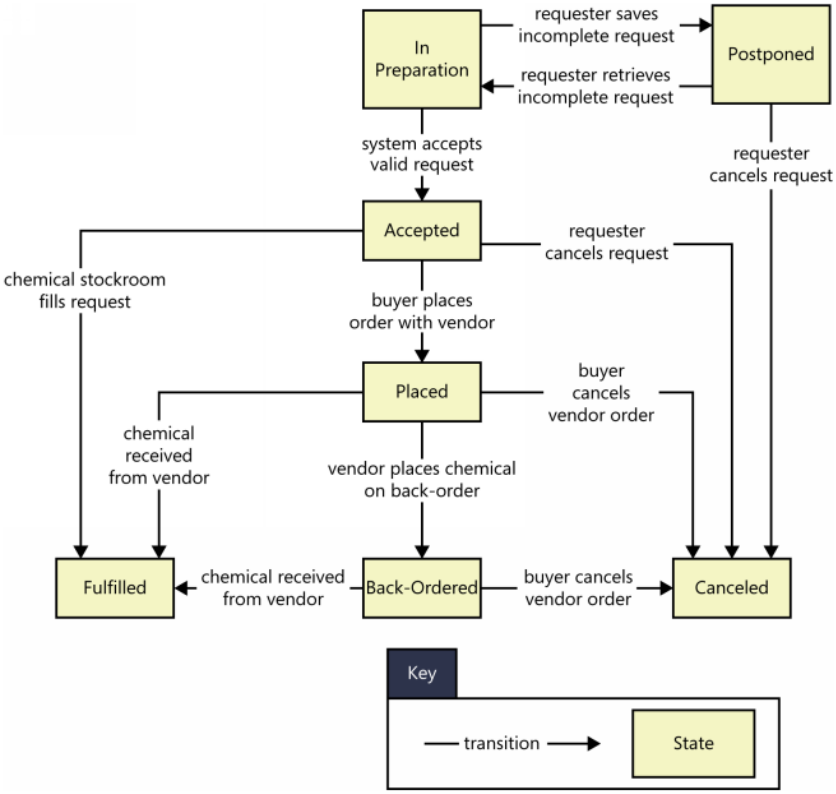
```
IF user is a LANDLORD
    IF valid rental details provided by the LANDLORD
        Rental Listing is added to the Rental Listings Data Store
        Rental is displayed on the front-end map interface
    ELSE
        Prompt the LANDLORD to input valid details
ELSE
    Reject
ENDIF
EXIT
```

- Decision Trees break down a process by working through the control (if-else) in a tree format, having questions denoted within diamonds, and outcomes in rectangles.
    - A decision tree must always have a distinct outcome and cannot look like a flow chart.
- Decision Tables list the conditions and actions on the left side, along with different permutations of outcomes on the right. Based on whether conditions are true or false, one action or another is noted as the outcome.
- State Transition Diagrams shows the flow of various states of the data in rectangles, along with arrows that are labelled with the events or conditions that change the state. There should be a start, idle, and end state indicated.
    - A state change can only take place when well-defined criteria are satisfied, such as receiving a specific input stimulus under certain conditions.
    - This example is what an engineer would use to determine requirements and triggers with a customer, while our UML-based State Transition Diagram would go into more details with events and would show activities.
    - The STD should only focus on the process that we are required to focus on in the question, and not expand outside of that process.



Requirement Number					
Condition	1	2	3	4	5
User is authorized	F	T	T	T	T
Chemical is available	—	F	T	T	T
Chemical is hazardous	—	—	F	T	T
Requester is trained	—	—	—	F	T
Action					
Accept request			X		X
Reject request	X	X		X	

- States are often labeled with a word ending in 'ing' (e.g. ordering, paying, reading).
  - Or past participle of a verb (e.g. returned, damaged, sold).
  - It's probably a good idea to come up with a list of possible states first, and then fill in the blanks in terms of the state transition (event) arrows.

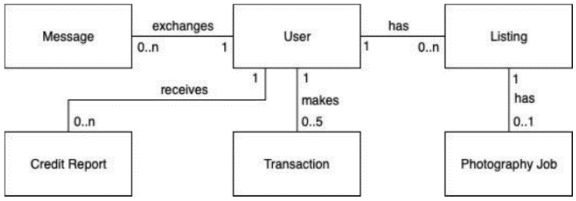


- Data Dictionaries are a collection of definitions for data flows and stores, in alphabetical text format.
  - Includes columns for field, type, length, PK, FK, and description (in his slides).
  - Collecting the information about composition, data types, allowed values, and the like into a shared resource identifies the data validation criteria, helps developers write programs correctly, and minimizes integration problems.
  - While primitives are represented by one entry in the dictionary, a data structure (or record) is composed of multiple data elements separated by plus (+) signs.
  - If multiple instances of a data element can appear in a structure, enclose that item in curly braces.

Data Element	Description	Composition or Data Type	Length	Values
Chemical Request	request for a new chemical from either the Chemical Stockroom or a vendor	Request ID + Requester + Request Date + Charge Number + 1:10{Requested Chemical}		
Delivery Location	the place to which requested chemicals are to be delivered	Building + Lab Number + Lab Partition		
Number of Containers	number of containers of a given chemical and size being requested	Positive integer	3	
Quantity	amount of chemical in the requested container	numeric	6	
Quantity Units	units associated with the quantity of chemical requested	alphabetic characters	10	grams, kilograms, milligrams, each
Request ID	unique identifier for a request	integer	8	system-generated sequential integer, beginning with 1

Data Modeling

- Data Modeling is used to define and analyze data requirements needed to support business requirements. It shows the relationships and structure of the data, ranging from conceptual to logical to physical data models.
- Entity Relationship Diagrams give a high-level static view of the data as entities (capitalized singular nouns and rectangles), which represent tables in a relational database.
    - Relationships are shown with a line connecting two entities (lowercase verb), and cardinalities shown ('0..*' meaning optional and '1..*' being mandatory).
    - The ERD should only include datastores from the data flow diagrams.
        - The names should all be capitalized.

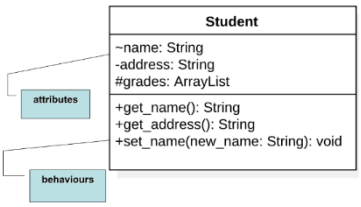


Software Release Planning

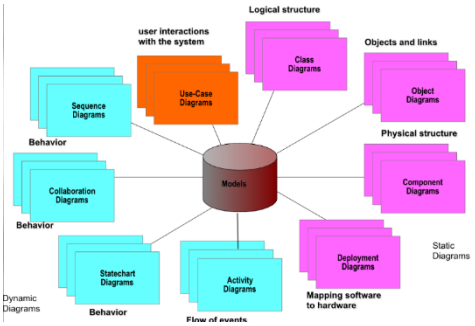
- Software Release Planning is a collaborative plan for delivering sequential increments of a software product, based on the customer's priorities. Simple voting, cost value, and systematic decision grids are approaches. Pairwise has a n x n matrix where each person votes for how one aspect (performance) relates to another (security), using 1 or 0. The number of comparisons is n(n-1)/2.
- The Eigen-Vector Approach to software release planning creates an n x n grid of symmetrical feature comparison values that is summed down the columns, normalized by dividing each element in a column by the column's sum, and then averaged across the rows to obtain the normalized principle eigen-vector (or priority vector).

Introduction to Object Technology

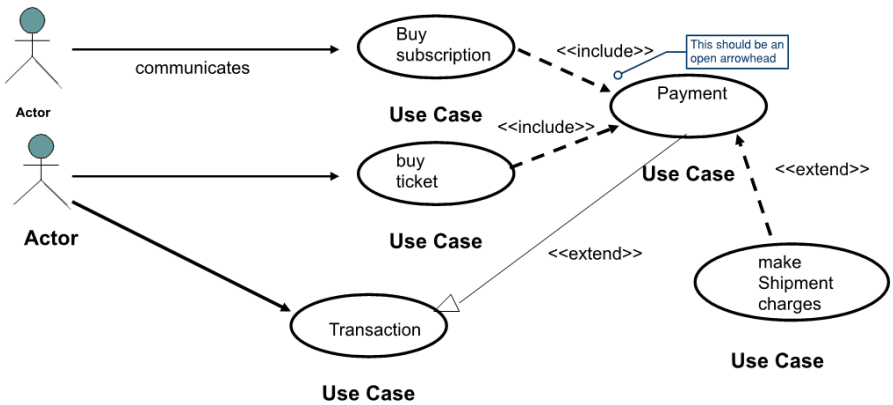
- An object is something tangible, conceptual, or something toward which a thought or action is directed.
    - Three characteristics of an object are state, behavior, and identity.
    - Two key components are attributes (representing state) and operations (representing behavior).



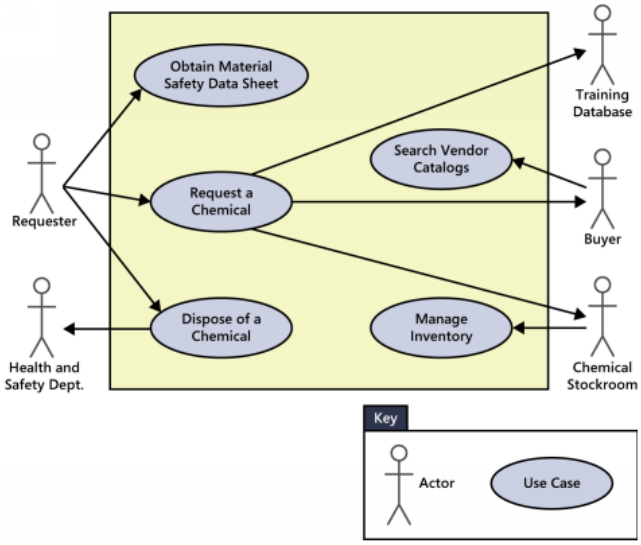
- UML (Unified Modeling Language) is a modeling language that methods use to express design.
    - Several models are used to give different view of the same system.
    - Behavioral models include: Use Case Diagrams, Sequence Diagrams, State Transition Diagrams, and Activity Diagrams.
    - Static models include: Class Diagram, Object Diagram (instance of Class Diagram), and the Component Diagram.



- A Use Case Model shows the system's intended functions (use cases) and its environment (actors).
    - An actor represents anything that interacts with the system, while a use case is a sequence of actions a system performs that yields an observable result of value to an actor.

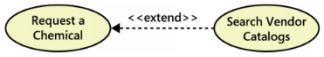


    - The actors shown in the use case should also be shown in the sequence diagram (consistency between the two).
    - The actors are connected by lines to the use cases that they carry out.
    - Any lines without arrows are assumed to be bi-directional, and that all lines to the actors are interfaces.
    - A use case describes what a system does, but not how it does it (not a flow chart).



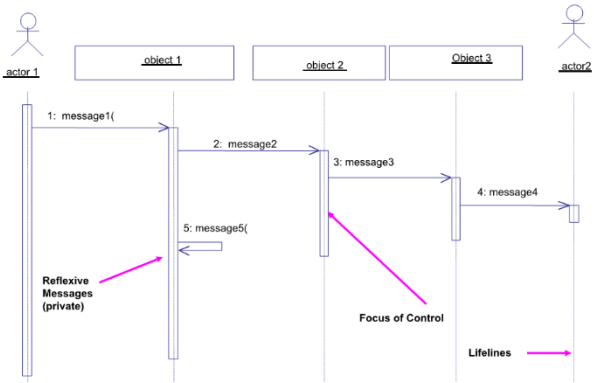
- It helps you to determine the functionality and features of the software from the user's perspective.
- A use case describes how a user interacts with the system by defining the steps required to accomplish a specific goal.

- Note that the use cases are placed in a rectangle, but the actors are not. The rectangle is a visual reminder of the system boundaries and that the actors are outside of the system.
- If we want a use case to be able to execute another function when performing their own function, we would use the <<extend>> keyword with a dashed arrow. It is optional if the use case uses the functionality of the use case it extends from.

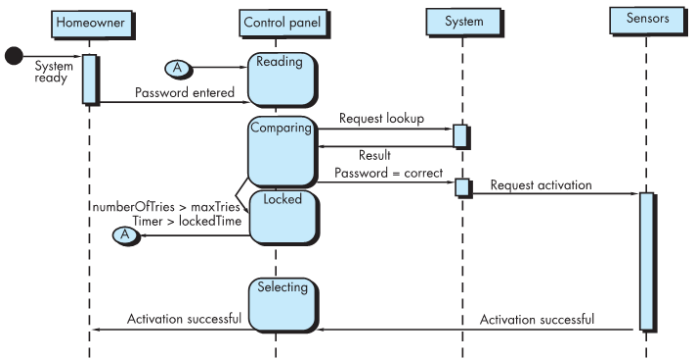


  - If we want a use case to not have to duplicate functionality, we would <<include>> the subordinate use case. This is analogous to calling a common subroutine in a computer program, and would be mandatory.
  - For arrow directions, remember that when we extend, we extend from something, and when we include, we include that item.
  - Note that these should only be used if there is a strong relationship between use cases, and is the only time that two use cases should connect to each other.

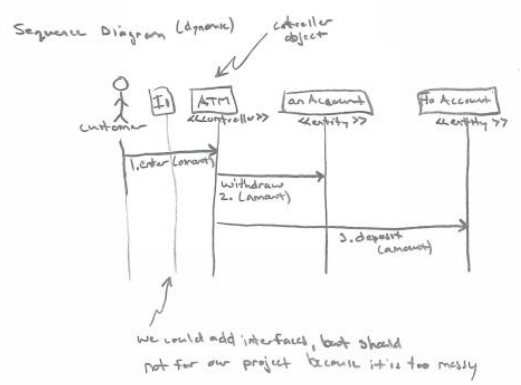
- A Sequence Diagram models the dynamic aspects of a system in UML, and we typically have one per use case.



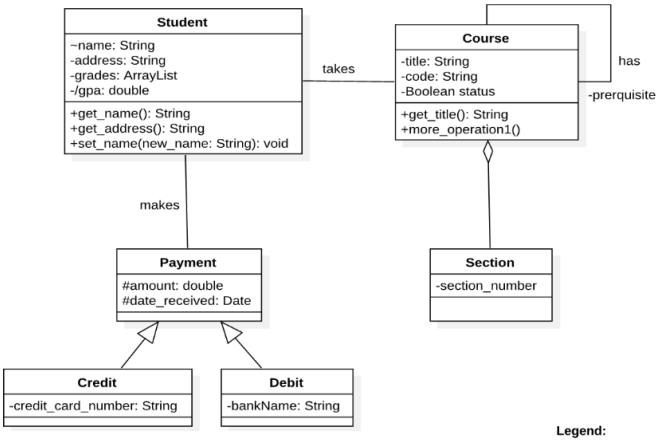
- It is an interaction diagram that indicates how events cause transitions from object to object.
- Shows the objects participating in the interaction and the sequence of messages exchanged.
- If a line (message) is going to an object, then that object has the operation (message).
- Once events have been identified by examining a use case, the modeler creates a sequence diagram – a representation of how events cause flow from one object to another as a function of time.
- Each of the arrows represents an event (derived from the use case) and indicates how the event channels behaviour between objects.



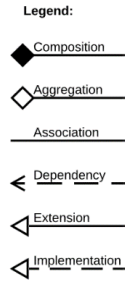
- Time is measured vertically (downwards), and the narrow vertical rectangles represent time spent in processing an activity.
- A black circle with an arrow coming from it indicates a found message whose source is unknown or irrelevant (for ENSF 613, we show the actor).
- You can show an object sending itself a message with an arrow going out from the object, turning downward, and then pointing back at the same object.
- You can show object creation by drawing a dashed arrow appropriately labeled (for example, with a «create» label) to an object's box. In this case, the box will appear lower in the diagram than the boxes corresponding to objects already in existence when the action begins.

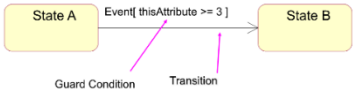


  - Once a sequence diagram has been developed, all of the events that cause transitions between system objects can be collated into a set of input events and output events (from the object). I imagine that these events should conform to the State Transition Diagram for an object.
  - Note that the actors shown in the sequence diagram should be consistent with the actors for that particular use case (shown in the Use Case Diagram).
    - As well, the classes shown in the Sequence Diagram should be shown in the Conceptual Model.
- A Conceptual Model (or Class Diagram) is a static view of a system that shows the collection of classes involved to build a system or subsystem.
  - Relationships could be shown between classes, similar to what we've done in ENSF 593/594 in the past.
  - Cardinalities (also known as multiplicities) should also be shown.
  - We must have consistency between the Conceptual Model classes and those shown in the Sequence Diagrams.
    - The Sequence Diagram should inform the classes that we show inside of the Conceptual Model and should be completed before the Conceptual Model (but after the Use Case Diagram).

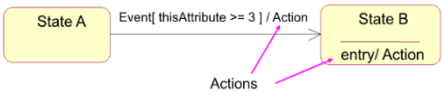


- State Transition Diagrams show the state of the machine which specifies the sequence of states that an object can be in, the events and conditions that cause the object to reach those states, and the actions that take place when those states are reached.
  - According to our notes, a STD could be used for either an object or process.
  - Look for possible states by evaluating attribute values, evaluating operations, defining the rules for each state, and identifying the valid transactions between states (typically ending with 'ing).
    - Note that the state of a class has both passive and active characteristics, where passive state would be the current status of all an object's attributes, while the active state would indicate the current status of object's undergoing a transformation or process.
    - An event (or trigger) must occur to force an object to make a transition from one active state to another, with the label of the transition arrow indicating the event that triggers the transition.
  - Also examine the sequence diagrams for the class being modeled, since the interval between two operations may be a state.
  - The initial state is the state entered when an object is created, and only one initial state is permitted (shown by a solid circle).
    - The final state (optional) indicates the end of life for an object, and more than one final state can exist (shown by bull's eye).
  - Events can use Boolean guard conditions (or expressions) to show when that event is eligible to fire (only when true), after which the transition from one state to the next will occur.
    - In general, the guard for a transition usually depends upon the value of one or more attributes of an object.
  - Actions are executable atomic computations that result in a change in state of the model, or the return of a value.

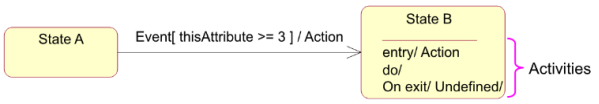


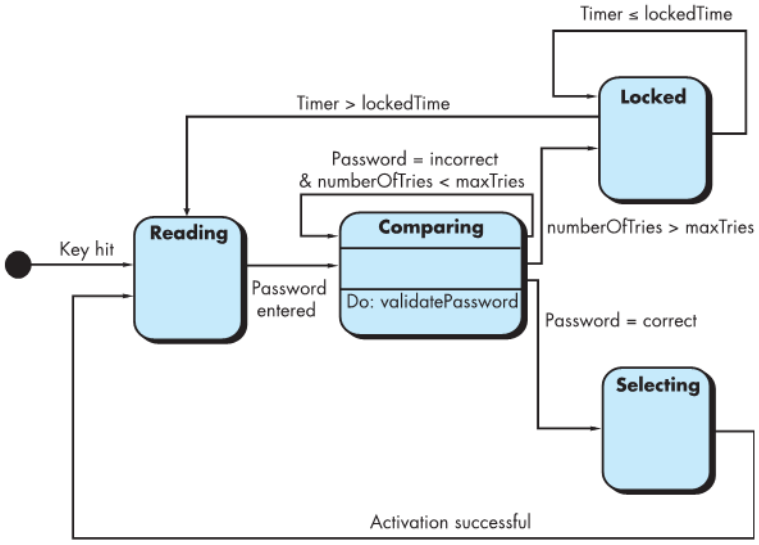


        - An action occurs concurrently with the state transition or as a consequence of it, and generally involes one or more operations (responsibilities) of the object.

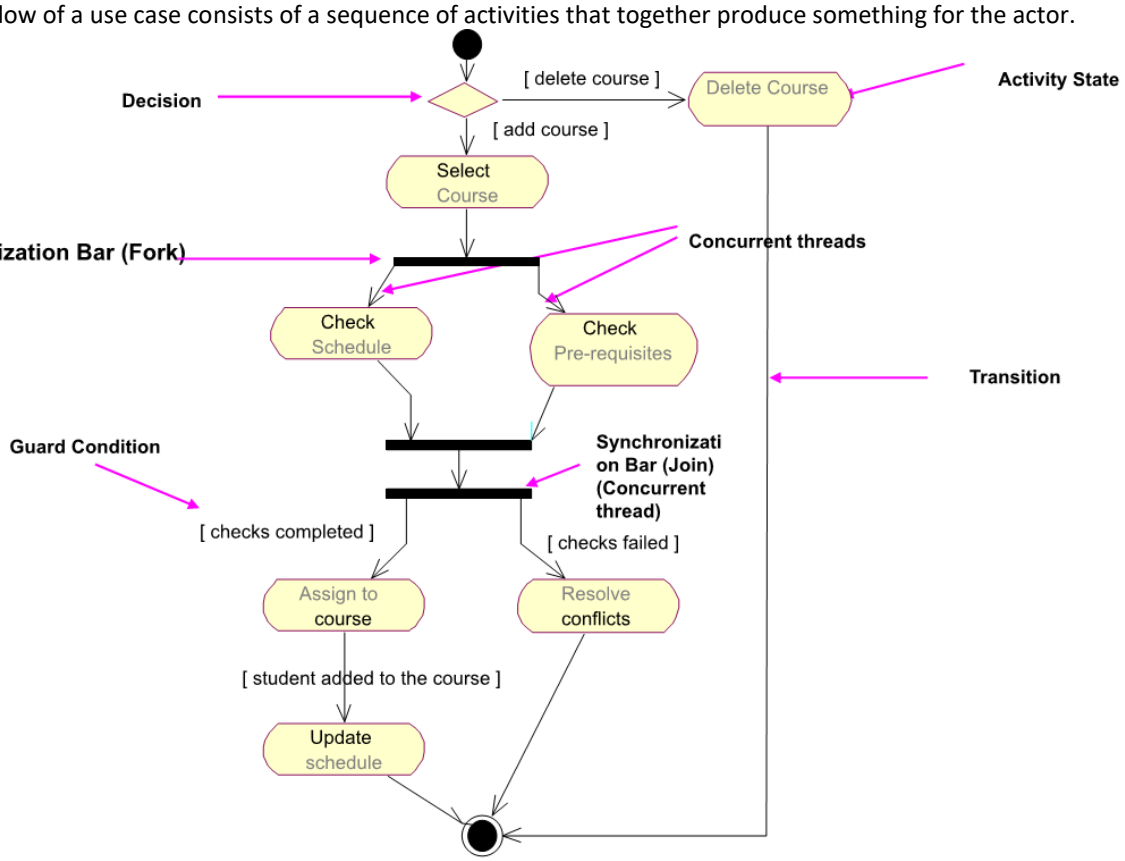


    - Activities are on-going, non-atomic executions within a state machine which take some time to complete.
    - An activity starts when a state is entered or can be interrupted by an outgoing transition.



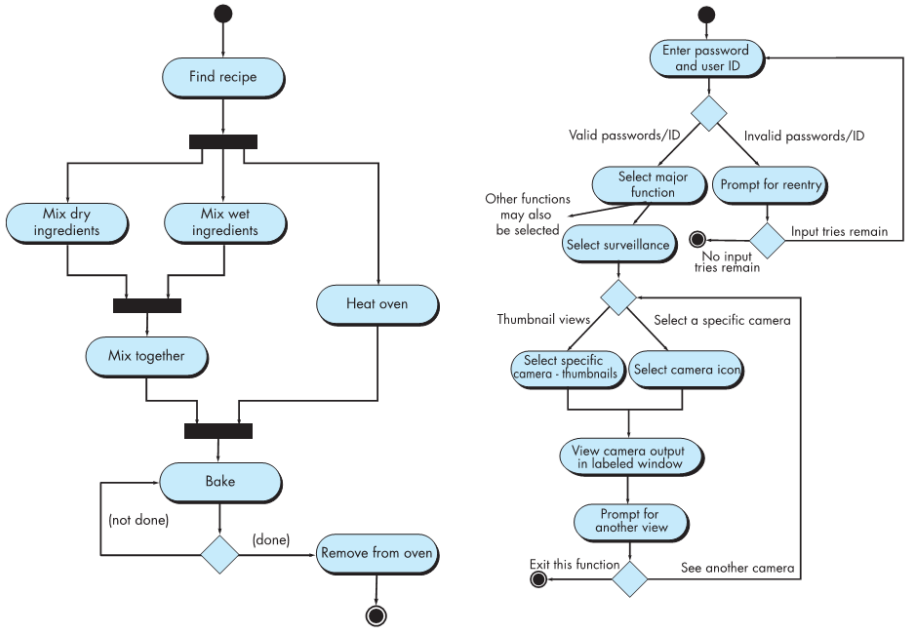


- An Activity Diagram also describes the dynamic aspects of the system and is like a flowchart that illustrates the flow from one activity to another.
    - The activity can be described as an operation of the system and control flow is drawn from one operation to another.
    - The workflow of a use case consists of a sequence of activities that together produce something for the actor.

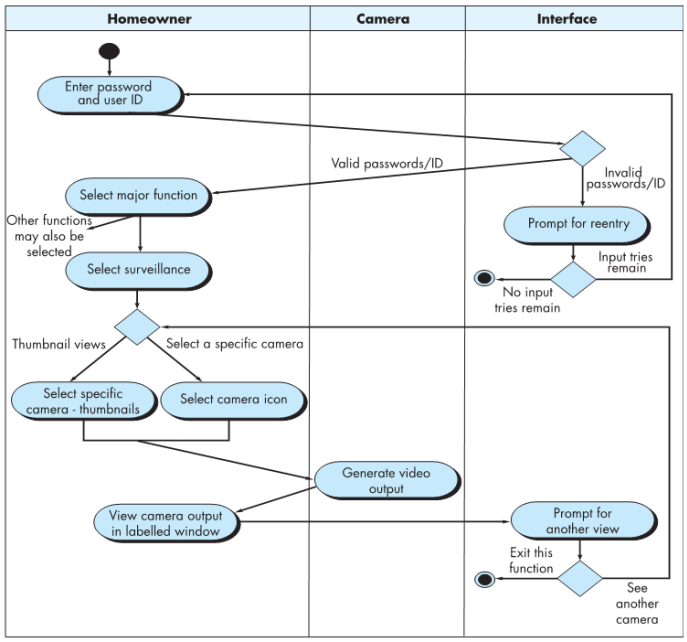


    - Similar to a flowchart, an activity diagram uses rounded rectangles to imply a specific system function, arrows to represent flow through the system, decision diamonds to depict a branching decision (each arrow emanating from the diamond is labeled), and solid horizontal lines to indicate that parallel activities are occurring.
        - Activity diagrams add additional detail not directly mentioned (but are implied) by the use case.

- Note that when a join occurs, flow cannot continue until all flows represented by incoming arrows have been completed.



- A useful variation is the Swim Lane Diagram, which allows for the representation of the flow of activities by the use case and at the same time, indicates which actors are involved.



- Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool.

Non-Functional Requirements

- Non-functional requirements define the overall qualities or attributes of the system (or quality attributes), which might be related to performance, security, usability, etc.
  - Can be mandatory or non-mandatory.
- Functional requirements are verbs, which non-functional requirements are attributes of these verbs.
  - Ex. The system should have a login, vs. the system should have a secure login.

Requirements Fit Criteria

- Fit criteria indicates whether a solution completely satisfies the requirements.
  - This should be expressed in numbers, and the solution should do no more and no less.
  - Using test cases and benchmarks is one way to measure, although some measures such as how 'nice' a product is will require creativity to evaluate.