model

* partial abstract representation of real world system
* inexpensive way to analyse communicate, test, and document understanding of system

types of models

* computational
  + computer simulations representing time-varying behaviour of a system
  + can be already out there
    - eg. Weather app.
    - getting data from the real world and then simulating the future (extrapolation)
    - can help to save money to not build more things
* analytical
  + mathematical models of relationships among variables in a system
    - eg. Banking/mortgage and estimating the liability of borrowing
    - use to determine another variable based on two other variables
* nonanalytical/descriptive
  + describe the components and their relationships (how they are connected) in a system
  + qualitative and not quantitative – no numbers are involved
  + don’t require a high level of expertise, easier to build and understand than the other two
  + two different groups
    - model data – like entity-relationship models.
    - Model the Application – like UML models
      * works on data/refer to the data
      * graphical notations to describe and design SW systems (esp. OOA)
      * notations and meanings are defined by computer industry standard consortium (OMG- object management group)
      * focus on modelling the logic and structure of the application that uses that data
      * also include sysML (general purpose SYSENG – use case, activity, sequence diagram) , BPMN (Business Process Modelling Notation flowcharts – not targeting SW, for stakeholders – domain knowledge and organizational systems)
  + UML notation divided into (UML is descriptive/qualitative)
    - UML Diagrams model
    - Structure (the static view of the system in terms of its components)
      * Class diagram
      * Component diagram
      * Object diagram
      * Composite structure diagram
      * Package diagram
      * Deployment diagram
    - Behaviour/dynamic (shows how the system and its components behave over time - workflows)
      * Use case diagram
      * activity diagram
      * state machine diagram
      * interaction (interaction within and outside of the system, interaction among the components of the system identified and between the system and external actors, how system acts with external entities)
        + sequence diagram
        + communication diagram
        + timing diagram
        + instruction overview diagram
* don’t need to draw all models to develop a system
  + analyze, communicate, test, document
* thinking and communication tools
  + to understand and agree with team
* model smartly
  + start rough, refine as needed, becomes a long-term asset for team – before they are transferred to the tools for long-term reference and updates – have long term value as knowledge asset for the team
  + Invest early so when SW grows it will help plan – (iterations of design and implementation will help with rough to refine)
* Computer-aided SWENG tools (CASE tools)
  + Helpful in tasks through SDLC: modeling, code generation, reverse engineering, analyzing code complexity and other metrics
  + Helpful when creating complex models and keeping them upgraded on ongoing basis
    - E.g. visual paradigm, MS visio, PlantUML, IBM rational rose
* Use Case Diagram
  + Precursor to use case specifications
  + Capture the overall functionality of a system – high level
  + Use notations for actors, use cases, and relationships among them
  + Used as a summary of all use cases in system
    - Are documented elsewhere in use case specifications
  + Key elements - Use cases (bubble verb/noun), systems, actors(user role in system – human or other system, primary (LHS, goal fulfilled, triggers use case) or secondary(RHS, involved in some way, often external (like a DB), interface with primary actor, can use same notation as primary or a box stereotyped with <<>>,), associations (btw actor and use case – any combination, use solid line arrowhead from primary to use case and then arrowhead from use case to secondary etc.…btw use cases
  + Associations
    - include – when functionality across multiple use cases, use to reuse e.g. Use login for all use cases that would use login – have a dashed arrow pointing to reusable use case (and the base use case depends on login/incomplete without it)
    - extend – use to attach an optional or additional behaviour to a use case (like select beginning and end dates to display schedule) keeps base use case unchanged as more functionality added – base use case is independent of the extend use case/optional to be added later
    - generalization
      * not as common, specialized form of another (more general) use case, can inherit a lot from more general use case, use solid line and hollow arrowhead
      * can also use between actors to show inheritance
    - can all help to find common functionalities available and identify overlaps help to clarify requirements, user acceptance testing, and acceptance criteria.
    - Also help with data modelling when data needs to be available to models
* Activity diagram
  + Behavioural diagram
  + Captures workflow or process model of a system
  + Similar to flow charts, but with parallel behaviour and multiple actors
  + Often drawn by users, business analysts, and developers to capture requirement understanding (complex scenarios in use case specifications)
  + Have a start and end node, rectangle to show the action, and an arrow between to show the flow. Fork – parallel flow – split – one incoming/multiple outgoing. join – parallel flow ending, joined, multiple incoming/one outgoing. Decision – diamond – yes/no (condition/guard)- one outflow but multiple options, and merge – multiple inflow, one outflow. Swim lanes – each actor gets a swim lane where they do their actions, can be vertically or horizontally.
* Class diagram
  + Used to model the types of objects and relationships among them.
  + Granular – lowest level of abstraction (classes and interfaces)
  + Widely used UML structural (STATIC) models
  + Often created by developers as design activity
  + Noun = class, property = attribute, verb = relationships
  + Classifiers – common abstraction, types of entities in system, abstract metaclass whose concrete subclasses classify different types of values, supertype of things, such as abstract (italicized or stereotyped), concrete class, interfaces (can stereotype), enums (stereotype), generic classes, active. Box with 3 compartments, name in top, attributes and then behaviours
  + Features – structural and behavioural characteristics of classifiers. Structural- properties or attributes in the middle. Behavioural - operations/methods observations in bottom
  + Relationships – how these entities are related to each other, depending, inheriting, implementing abstract classes. Associations (aggregate/empty diamond or composition/filled diamond), generalizations (inheritance btw classifiers unfilled arrow with solid line pointing to parent class) and dependencies (arrow pointing to depended on class with dashed line).
* Write use case and basic scenario (steps in process of use case), write nouns for classes/classifiers (roles), identify attributes (names), relationships (behaviour – between the classifiers, responsibilities or things done by classes so other classes can use them also, look at the way class interacts with another class to develop methods in an “agile approach”)
* Interaction diagrams – model collaboration among objects in system
  + Sequence diagrams
    - Most common uml/interaction diagram. DYNAMIC
    - Show behaviour of an interaction among two or more entities in a scenario
    - Help identify behaviour we need to implement in the code
    - Read top to bottom (flow of time) and left to right (flow of messages – message sent to participant (ask them to do something) – solid arrow to the right, participant response - dotted arrow to the left)
    - Lifeline dashed line vertically with activation bars (indicated the object is waiting for the interaction to be completed after response sent and alive on lifeline)
    - Participants (objects) at top in rectangle (named objectA:ClassA)
    - The messages sent are the behaviours of the classes back and forth
    - Fragments
      * Conditions: using opt (option – if condition, condition needs to be true for user to be in box and exchange of messages in box), alt (alternatives – if/else condition, one or more conditions if first condition isn’t true)
      * Loop – block of messages that are passed as long as condition is true.
* Sequence diagram and class diagram developed “iteratively” back and forth until you can code and develop test cases, brings clarity and alignment with what is expected
  + Communication diagrams
    - Similar to sequence diagrams, but NO LIFELINE
    - Focus on objects are linked to each other through messages
    - Sequence of messages are shown by numbering the messages
    - More difficult to read sometimes
* Object diagram
  + To model example configurations of instances in a system (class diagrams show relationships among entities throughout the life of the system while running)
  + Help visualize what systems state/objects in a system would look like at a particular point in time
  + Structural aspect of system and not behavioural (no methods)
  + Represent their instances, also known as instance diagrams
  + Similar to class diagrams (can be derived from class diagrams)
    - Objects (that are instances of a particular classifier)
    - Attributes or instances variables with their values, also called slots
    - Relationships (among various objects alive at that particular time)
  + Naming objects
    - InstanceName : ClassName (Name is also underlined to show it is an object), name be named class name only – but need to retain colon (e.g. :ClassName)
  + Slots or Attributes
    - Don’t need to show the values of all attributes of object
    - Only need to show the relevant to what you are trying to model
  + Relationships
    - Only draw arrows if relationships between instances is important – can just draw a line
    - Similar to what is in class diagrams
      * Association - directed association, bidirectional association, aggregation, composition
      * Generalization
* State machine diagram (also known as state transition diagram, state chart diagram, state diagram)
  + Behavioural diagram, models different states of entity and transitions in a system.
  + Protocol – model interaction sequences, more specialized – emphasize the sequence of the transitions
  + Behavioural – model even-driven behaviour of an object
    - More widely used
    - State
      * represents a single state of a single entity being modelled
      * often an instance of a class or an object
      * include name and internal behaviours and transitions
      * composite state – models hierarchy of states by nesting states within a higher abstraction of state
    - Transition
      * To model movement from one state to another
        + Arrow from source to target
        + Trigger[guard]/activity

trigger is something that happens outside of the object being modelled

guard is any condition that may be applicable for transition to happen

activity takes place when transition happens

* + - Regions
      * Fragments of state diagrams that run concurrently
    - Vertices
      * Abstract term for node that are source of target of transitions, such as states or pseudostates
      * E.g start, idle, in use, stop
* Component diagram
  + Model systems of autonomous reusable modular units with interfaces “wired” together to form a system
  + Only relevant when you are developing and using reusable components
  + Based on component development (CBD) approach
  + Concepts and ideas similar to class diagrams
  + Key elements
    - Components
      * Self-contained unit of SW that can be fit into a larger system and can be replaced at design time or runtime by a component that offers equivalent functionality
      * Need to show provided and required interfaces, see how different components use and offer functionalities to make larger system work
      * Has a stable interface specification
        + Any piece of SW that meets the specifications, can play the role of that component
      * Notation – name in rectangle with two small rectangles on the left, or the rectangle with two small rectangles on the left at the top corner of a larger rectangle and the name in the middle
    - Interfaces
      * Define the component being modelled/can do
      * Provided interfaces – lollipop notation
        + Interfaces implemented by the component
      * Required interfaces – socket notation
        + Interfaces that must be implemented by others to use the component
    - Relationships
      * Lines connecting various components
        + Provided or required interfaces
* Package diagram
  + Model the structure (not behavioural, no interaction) and organization of a large object-oriented system
  + Package elements from other diagrams
  + Useful when building a complex/large application with 100s of classes
    - Help group them together, create hierarchies, and show relationships across these groups and hierarchies
    - Classes are related to each other
    - Help abstract the details and focus on how big parts of the system fit together
    - Package
      * Models a namespace for its members – can be a unique name
      * Helps bundle together various elements of system into hierarchies and groups/Helps group related classifiers in a hierarchy
      * Members inside - Can show all nesting of packages (classes) independently
      * Members hidden – just show the package name on the folder without class names
      * Members outside but owned – package name on folder and classes outside
    - Relationships
      * Among the packages
      * Dependency- shown with arrows to show one package dependent on another
      * Stereotype
        + Merge

Dependency relationship

Contents of target package are combined (merged) into the source

Similar to generalization

* + - * + Import

To model other forms of dependencies such as generalization and associations among elements across packages, another dependency relationship

* Deployment diagram
  + How the SW system is laid out, to model execution architecture of a system
  + Captures how the SW and HW components are laid out
    - Nodes
      * Represents a deployment target that can host SW (what it is installed on – HW device or execution environment)
      * Shown as a cube with name at the top
    - Artifacts
      * What gets deployed on node
      * Rectangle with a file icon on top right side
      * E.g. executables, documents, html files
    - Relationships
      * Communication paths – most common and important
        + Solid line from one node to another node
        + Indicate nodes talk to each other when SW executed
        + Are bidirectional
      * Dependency
        + Between nodes and artifacts or artifacts themselves
        + Same meaning as class diagram
* Interaction overview diagram
  + Help model overview of control flow
  + Specialized mix of activity and interaction diagram (sequence or communication diagram)
  + Shows some interaction inside an action node
  + Not popular
  + Similar notation to activity diagram (uses stop, decision, merge, join, and directional arrows)
  + Different – action node replaced by 3 notations
    - Frame – rectangle with header at top left, indicating something about what action it represents
    - Node
      * Interaction
      * Interaction use
* Composite structure
  + Decompose/model the internal structure of a classifier or structural element (class, component, or object) in the system
  + Focus is to model how different elements come together when the system is running
  + Similar to component diagrams, but help model what is inside the components
  + Classifiers
    - Structural entities/elements whose internal structure of a system is being modelled, e.g. A class, object, or component
    - One component could be nested inside another, like other complex systems
    - Can be shown as a component or shown as a class with internal parts (and not attributes and behaviours)
  + Parts
    - Elements/internal elements inside the structure classifier being modelled
    - Often composition (association) elements with the outer classifier
      * The part is embedded inside the classifier and not a shared element with any other classifier uses solid arrow
    - If an aggregation (association) and the element is shared elsewhere, use a dashed arrow to show “porous boundary”
    - If more than one of part, use a number in the top right corner
  + Ports
    - External connection ports, classifiers interaction with its environment
    - Noted as small square
    - Connected through dependency link internally with parts of system
  + Connectors
    - Communication links among various parts of the system
* Timing diagram
  + In electrical engineering when the states of various components in a circuit are mapped
  + Model objects changing states along a timeline
  + Special form of interaction diagram
  + Interaction diagram to show how states of different objects change over a period of time
  + Changes could be triggered by clock time, or a message sent by another object
  + Timeline
    - Scale to mark clock time, often relative, in any unit (seconds, minutes etc.)
  + Frames
    - Creates space for the objects lifeline to be modelled, represents the object whose lifeline is to be modelled
    - Name in top left corner
  + Lifelines
    - Dynamic part of diagram, represents how the state of object is changing over time
    - Value lifeline to show change (e.g. on/off)
    - State lifeline - Flow that has discrete values (e.g. State is on or off)
  + Messages
    - Interaction among objects being modelled
    - Help to model interaction of the system
    - A label to indicate what happened or changed