Final Exam SG

Thursday, April 19, 2018 6:40 PM

Topics:

- Systems Analysis
- SDLC
- Scrum, Agile, Teams, User Stories
- Clients, Requirements Elicitation
- Functional Objectives
- Events, Event Table
- DFD Notation, Context Diagram
- UML, Use Case Diagram
- Use Cases and Scenarios
- Prototypes
- Events and Things, Object Thinking
- Class Modeling
- Object Interaction Diagrams, Sequence Diagrams
- State Charts

Intro to System Analysis

- A system is many things:
    - A word which allows us to talk about things we do not know anything about.
    - An organized set of doctrines, ideas or principles, usually intended to explain the a of a systematic whole.
    - A system is any set of interrelated, interacting components that function together objectives.
- Systems have problems:
    - Adaptability, as some are more specialized and cannot adapt to every situation
    - Maintainability, as larger systems require more maintenance
    - Systems within systems, as systems can be complex as they encompass and reside
    - Systems grow larger over time, adding complexity
- Software Dev is labor intensive and has a high failure rate. Maintenance changes are a p leads to problems involving flexibility (not being able to respond to change), and resilien handle changes during maintenance).
- Systems Dev can be complicated, with managers not knowing software complexities and

arrangements or working

r to achieve specific

e in other systems

problem, and poor design
nce (not being able to

d consultancy being

expensive.
- Flexibility is the ability of a system to handle unforeseen events or transactions. Resilien to handle changes DURING maintenance without other problems happening with every
- We use System Analysts to clear up requirements and clear up what parts will go into ou
- Analysis is to break down, understanding and documenting business needs. We focus on do. Design is to build up, choosing among alternative implementations. It focuses on ho requirements.
- An analyst is and has many things:
    - They solve business problems.
    - They have technical knowledge and skills, like hardware and software, spec design techniques for completing dev activities. They also have business knowledge and functional work process and org structure.
    - They can model things, documenting what we learn and communicating that with techniques so we can model things properly.
    - They have people sensitivity, interviewing and listening to people and gathering a

SDLC

- The System Development Life Cycle describes the flow of tasks that must be completed functionally correct system.
- Correctness defines if a system meets its specification. We ensure this through two acti
    - Verification, were we ask if we are building the system right.
    - Validation, where we ask if we are building the right system.
- Traceability follows the SDLC forward or backwards, ensuring nothing is missing or no fe respectively. That flow is Requirement to Design to Implementation forwards. Forward nothing is missing, and goes from requirement to design to implementation. Backwards way, and ensures that no features are undocumented.
- SDLC phases are groupings of activities, SDLC stages are steps ordered in time.
- The phases are:
    - Requirements Identification, which ID's the problem to be solved, the features of case and the accepted criteria.
    - Analysis, which is where we understand the domain, understand the problem, an
    - Architecture and Design, where we determine the overall structure of the system components and how they fit together.
    - Implementation where we build the system
    - Testing where we check if the system is working
    - Deployment where we put the system to work
    - Maintenance where we keep the system working and evolve it to keep it useful
    - Project Management (which occurs during all of the above phases), where we pla

nce is the ability of a system
little change.
ur system.
n what the system should
w the system meets its

n and analysis, and
skills, working with

n users, and have

ll needed info.

to design and implement a

ons:

eatures are undocumented
traceability makes sure
traceability goes the other

the solution, the business

d understand the solution.
, and determine the

n, organize resource lead

control and coordinate the project.

- The stages are:
    - Inception, where we create an approximate vision, a business case, a scope, a hig high level estimates.
    - Elaboration where we refine everything above and do most of the requirements i
    - Construction where we build features
    - Transition where we deploy
- A software process is a well-designed set of partially ordered steps intended to reach a adding processes to reduce the risk of not meeting customer expectations. It helps achi
    - Repeatability
    - Predictability
    - Traceability
    - Improved quality using standardization
- Two approaches to SDLC are Agile, where we run all of the phases and stages in short b (the good way) and waterfall, where we run a single giant cycle.

Scrum and Teams and User Stories

- Scrum is a refinement of Agile. It features iterations and releases, and has a prioritized b
- It does all the requirements, design, code and testing all at once, little at a time.
- It values individuals and interactions over processes and tools, working software over co documentation, customer collaboration over contract negotiation, and responding to ch plan.
- Scrum can increase productivity, is the opposite of the waterfall approach, and reduces
- The team size is usually 5 - 10 people. We scale teams to application type and project du
- Scrum has many roles:
    - The product owner, who is responsible for maximizing the ROI of development, re backlog, accepts/rejects product increments, considers stakeholder's interests an requirements questions
    - The Scrum Master who facilitates the scrum process, resolves problems, creates a to team self org-ing, and has no management authority over the team
    - The Dev Team, who build, test the code all in the same room
- The Scrum Cycle starts with building the iteration backlog during the iteration planning from the product backlog and estimating them using story points. The sprint goes on fo daily standup meetings occurring answering the questions of what was done yesterday, and if any blockers exist. Builds and tests are run continuously during this time. Then, th finished increment of the product is approved or denied by the PO. The team then has a went right, what went wrong, and where things could improve.
- A pig is a team member who is committed to the success of the project. A chicken is son

h-level potential arch, and

dentification

goal. We follow cost
eve:

ursts over and over again

backlog of work.

omprehensive
hange over following a

time to benefits.
uration.

eprioritizes the product
d is the final word on

an environment conducive

meeting by pulling things
r a week to a month, with
what will be done today,
he demo occurs where a
a retrospective on what

meone who is interested

but not committed.
- A sprint planning meeting is where backlog items are negotiated for who will work on th team pulls backlog items and breaks them into sprint tasks.
- The product backlog contain all the requirements for the project. It is a list of all desired expressed as a list of user stories.
- The sprint review is where the team presents what it accomplished during the sprint. It prep time rule. The whole team participates. It is a demo, not a report, and the PO mark
- It has a couple of artifacts:
    - The product backlog includes everything that is known and validated that will go i
    - The sprint backlog, which is built at the planning meeting and each story is scoped
    - Burndown charts, which show how many story points were completed over the sp
    - The backlogs can contain requirements, defects, change requests, enhancements,
- User Stories are expressed as "As a type of user, I want some goal so that some reason"
- Epics are large user stories that can be decomposed into smaller stories that are easier t developed in parallel.
- A feature has many definitions:
    - An aspect, or element of a product
    - A cohesive bundle of externally visible functionality that aligns with business need
    - An Action that yields a Result on an Object.
- Features can be functional, or parts of the application that an owner specifically acts tha nonfunctional, which define things like security, uptime, and throughput of the system t based on your own design.
- Acceptance Criteria is a way to test user stories on the code base. It is how we verify rec template we have a couple of parts:
    - Given a precondition AND another one OR this one right here
    - When a user takes an action OR a user takes another action AND a user takes a pc
    - Then ensure something happens with the system AND ensure that something else else happens.
- INVEST describes attributes of good user stories:
    - Independent
    - Negotiable
    - Valuable
    - Estimable
    - Small
    - Testable
- Large user stories are generally known as epics. You can break down vague requirement from a user standpoint. This is done as large stories take longer to deliver than the equiv stories, which themselves are easier to understand and can be done by many more peo

hem during sprints. The

l work on the project,

is informal, with a 2 hour
ks done items during it.

nto the product.
d using story points.
print, and at what times.
, and technical debt.
.
to understand and can be

ds

at the product does, or
that tend to be created

quirements. In the Gricken

pintless action
e happens OR something

ts into more strict desires
valent amount of smaller
ple at once. You can split

epics by the happy path versus the alternate/exception path through them.
- Story writing is done near the start of the agile project, focusing on the when who why a
- The product review meeting is a meeting with the client to demo code. It is used to eval mutual understanding of the requirements, and to show off your work.
- The Myers-Briggs personality type classification is actually a load of pseudoscientist bull it anyways. Carl Jung made it, but Carl Jung's major influence was Sigmund Freud and th everything you need to know basically.
- It "allows people to make sense of the world through a series of psychological frames". dimensions:
    - Intuition or Sensing - How do we take in info, do we use details and facts, going fr abstract (sensing) or are we big picture oriented, and act on hunches (sensing)?
    - Feeling or Thinking - How to we act on info, are we logical, reasoning, and objectiv oriented (feeling)?
    - Perception or Judging - How do we operate in the world? Judging, where we are r routine, or Perception, where we are open and curious, willing to change.
    - Introversion or Extraversion - How do we interact with the world. Where do you g your bedroom (inner world, introversion) while not interacting with anyone, or sit accosting strangers for information (extraversion)?
- Tuckmans Model follows these steps:
    - Forming, nice ta meet ya
    - Storming - grr angery
    - Norming - ok ur cool lets lay down some groun rules
    - Performing - hey we good at dis

Client Requirements - Needed Questions and Fact Finding

- To get requirements out of clients, we usually have to define a couple of things:
    - The who, or who the stakeholders and users are
    - The what, the capabilities and features of the product
    - The how, or fact finding regarding the above two
    - The why, where we analyze and prioritize product features from above
- A stakeholder can affect or can be affected by an organization's actions. They have an in are stakeholders who use the system we make. They can fall into different groups with them personas to make the groups seem realer.
- Good ways of fact finding include collecting stakeholder surveys (the cheapest method) documentation, conducting interviews with users (expensive but effective), observing e
- We can guide an investigation using:
    - Initial General questions, like what are the processes, who are the users, and wha the system

and how of things.
luate your and the clients

shit but this class teaches
hat should tell you

It classifies on four

om the concrete to the

ve (thinking), or people

igid, decisive, and love the

get your energy? Sitting in
tting on a park bench

Exclusive Gateway - without Marker

Exit to exactly one path

Exclusive Gateway - with Marker

Inclusive Gateway

Merge when all inputs are complete

Parallel Gateway

Split and start all exit paths

Complex Gateway

Some other merging/branching

Event-Based Gateway

Pick path based on events, not

nterest in the project. Users
different needs. We assign

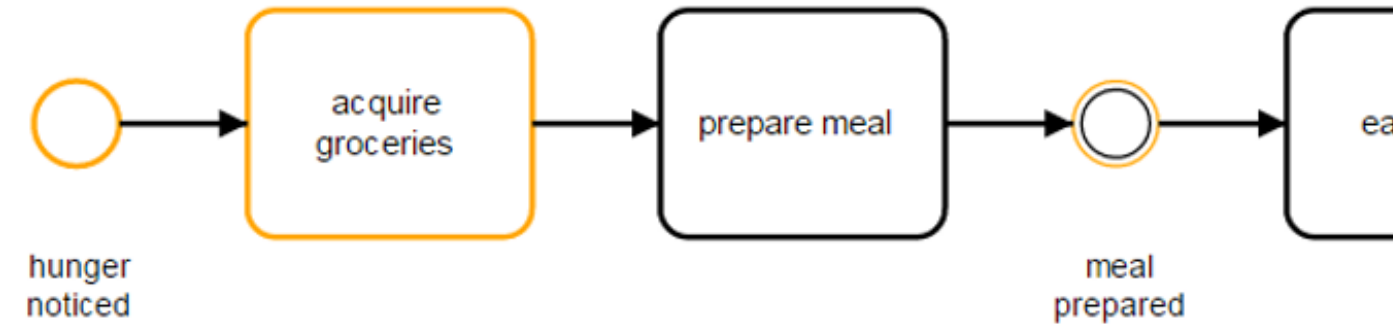
, reviewing existing
xisting business processes.

t do they want to do with

- More specific questions, like can we organize these activities the users want to do processes, and what information lives within those groups

Business Process Model Notation

- BPMN is a graphical representation for specifying business processes in a business proce
- There are a couple of types of objects:
  - Flow Objects, which are events, activities, and gateways
  - Connection Objects, which are sequence lines, associations, and message lines
  - Swimlanes
  - Artifacts

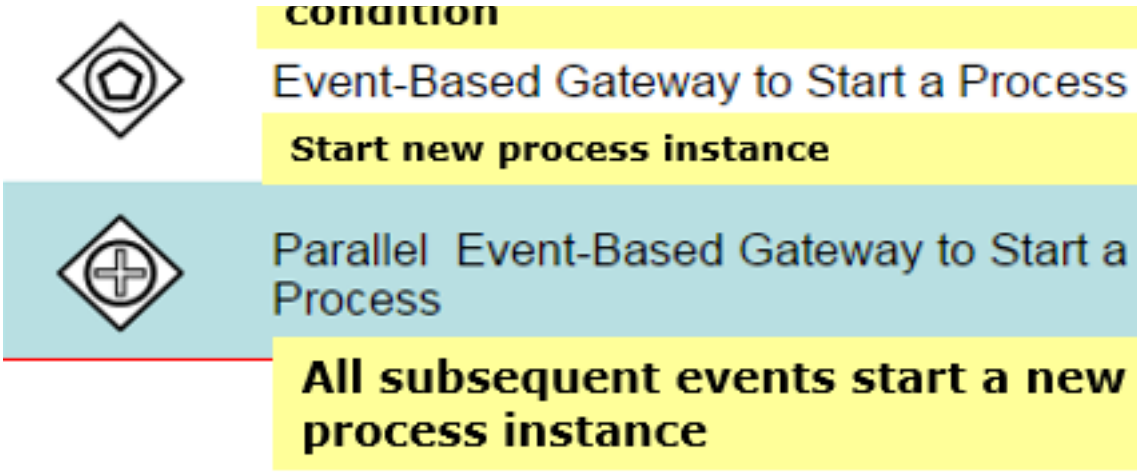


- Process triggered by Hunger
- Grocery and Food Prep activit needed
- When Meal Prepared event oc Eating activity is triggered

- Activities are labeled verb + object, events are labeled object + verb.
- Gateways are used to depict a split or merge of flows, with activities that determine the before a diverging gateway.
- Pools represent responsibilities for activities in a process. We split pools into lanes. Pool organization, role, or system in a business process.
- Sequence Flows/Lines are used to show the order that activities will be performed in a p

o into groups of related

ess model.



event
ties

ccurs,

e diverging conditions just

ls generally represent an

process, they cannot cross

sub-process boundaries, and cannot cross pool boundaries.
- Message Flows/Lines are used to show communication between participants, and canno
the same pool.
- Boundary Events must have at least one outgoing sequence flow, as they are things that
influences on the pool. They cannot have any incoming sequence flows.
- We can use a parallel gateway to merge all completed events before it before flowing to

Prototyping

- A prototype is a non-working mockup of the real system. We can afford to trash it, and
elicitation tool.
- It is used in Analysis as a way to validate requirements. This is in the discovery type of p
types of prototypes, the prototype becomes the actual system.
- We prototype in analysis to focus on content and business events, validate requirement
requirements.
- The prototype is usually a sim of the UI, which is everything the end user comes into con
system. Prototypes usually are more approachable than abstract models and reveal hidd
evaluate the feasibility of the new tech with the sponsor, and is effective for exploring t
and web.
- Low tech proto involves pen and paper drawings.
- Medium tech involves word processors or paint tools.
- High tech involves specialized prototyping tools that may evolve into the finished syster
developmental prototyping comes into play.
- To do discovery proto, have user review sessions where all classes of users review and a
prototype. Make notes about the problems they encounter.
- Don't put too much effort into a prototype, and don't go design crazy. Just focus on the

Functional Objectives

- Nonfunctional requirements are broad capabilities of the system. These involve reliabili
more. They also set constraints on the system, like time, cost, standards, and hardware
- Functional Objectives are formal expressions of user stories as system functions. These
positive statement of what the system will do. We justify each objective with its busines
make it measurable in some way involving something like money or time.
- These FO come from stated needs and problems and opportunities. A problem in this ca
regarded as unwelcome or harmful and needing to be dealt with and overcome. It is the
desired behavior in a system. An opportunity is something new, like tech, products or se
yet.
- FO's state what the system will do, why the system will do it, and how to know the obje
talking about the what, turn negative statements into positive ones, and describe the sy

ot connect objects within

t respond to outside

o the next event or activity.

is used as a requirements

rototypes. In development

ts, and discover missing

ntact with while using the
den issues. We do this to
arget environs like mobile

m. This is where

ttempt to use the

requirements.

ty, usability, security and
limitations
are typically expressed as a
ss purpose (IR AC IS) and

ase is a matter or situation
e gulf between actual and
ervices that do not exist

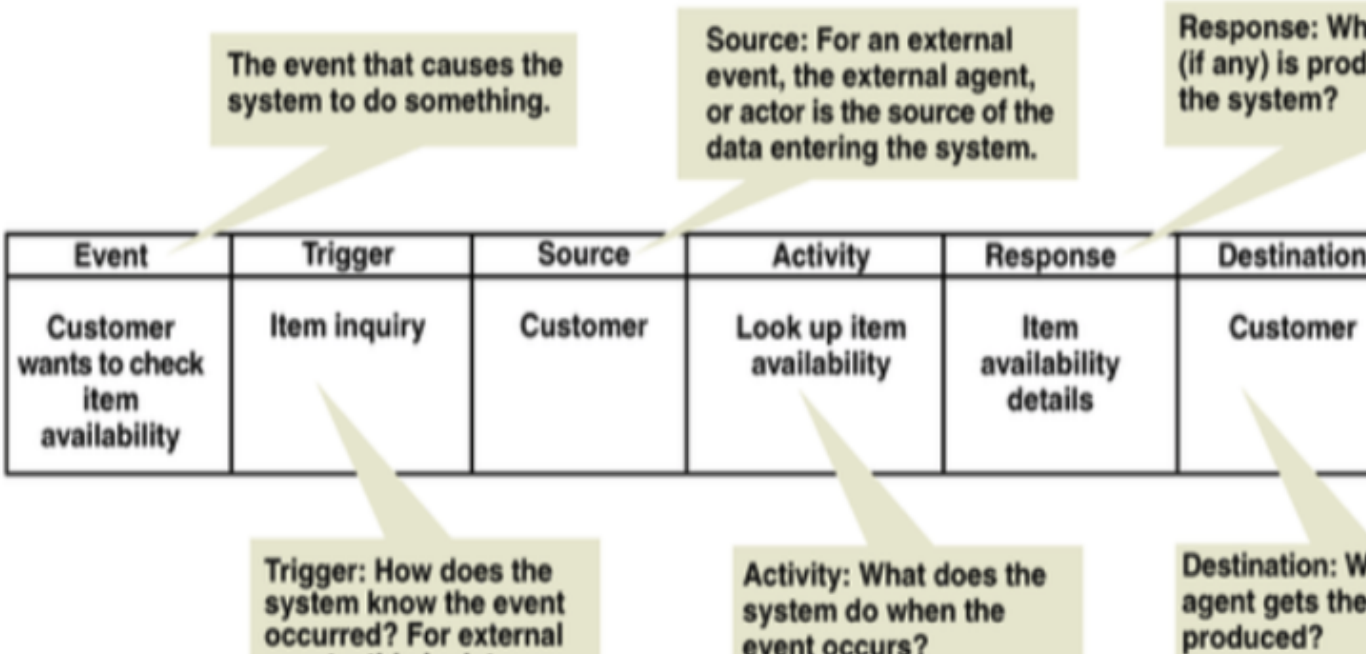
ctive has been met. When
ystem function using

externally viewable behavior. When you are stating the why, as IR AC IS, or if it will incre
and improve service. To measure if the objective has been met, put a number to increas
hours, improve x hours of response time, or other measures.

- The template is:
  - The system shall <do a function>
  - Creating <an IR AC IS benefit>
  - And the evidence of success will be measured by <a measurable variable>.

Events

- Analysts describe information system requirements using a collection of models. If you
you'll need more than one. Each model will represent some aspect of the system being
- Modeling has some benefits:
  - You learn from the process
  - You reduce complexity by abstracting out parts of the system
  - You remember details better
  - You communicate within the team and with many of your stakeholders
  - You have great documentation for future reference
- An event occurs at a specific time and place. It triggers all system processing. Requireme
relevant events and decompose systems into manageable units.
- There are different types of events:
  - External events, which are outside of the system and initiated by an external agen
  - Temporal Events, which occur due to reaching a point in time, and are based on s
  - State events, which occur when something inside the system is triggering a need f
- An event table can help draw the context and use case diagrams

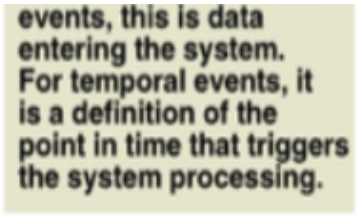


ease revenue, avoid costs,
se in price, decrease in

have a complex system,
built.

ents definitions determine

nt
ystem deadlines
for processing



- A trigger is how the system knows an event has occurred. It can be caused by data, like
  or by time, like when the first of the month rolls around.
- The activity is the process the system will perform. The trigger data gets transformed in
  internal data changes in the system.
- The response is what the system creates from the activity and trigger. Some events hav
- The source is who creates data for the event
- The Destination is who gets the response from the event
- In an expanded scope event, terminators are the ultimate source/destination of the dat
  those terminators can just be handlers of the data.

CRC

- The UC - FO - UC mapping maps user stories to functional objectives to events to use ca
- A use case documents the activity for an event. Each event will become one or more us
  done. You can then map this to the classes in your class diagram/CRC cards.
- There are many different types of Things. These things have relationships between each
  restrictions on the relationships. Relationships are naturally occurring association amon
  occur in two directions. The cardinalities are:
  - Zero to many, meaning the relationship is optional, it doesn't have to exist
  - One to one, meaning the relationship between two things is mandatory and cann
    and a new different thing.
  - One to many, meaning the relationship is between one thing and 1 to many more
- Things have attributes and values. The attributes are variables that describe the thing, a
  values of those attributes. More specifically, an attribute is one specific piece about a th
- Objects do work in the system and store information, and have attributes and behavior.
  classes, specific things are called objects, behaviors of objects are called methods.
- Actors and objects are things in the system. Actors are external, and objects are interna
  in the real world. Both send and receive messages.
- A better way to think about the class/object split is that Kal defines the class name as th
  that class as an object.
- OO Analysis models are the class diagram, which model the problem domain objects, ar
  which models the interaction.
- OO Design models are interface objects that make up the GUI, which allows users to int
  models.
- CRC Cards were originally a teaching tool for OOP. They are a method to gather and def

when an actor inputs data,

to response data, or

e no external response.

a. In a reduced scope,

ses.
e cases by the time you are

n other, with cardinality
g specific things. They

ot be between one thing

things.
nd the values are, well, the
ning.
. Types of things are called

l. Objects represent things

e class, and an instance of

nd the sequence diagram,

eract with problem domain

ine the user requirements

CRC cards were originally a teaching tool for OOP. They are a method to gather and def
for an OOP app. They are used to document the essential properties of a class.

- Each CRC card in the model represents a class in the solution, and a collection of CRC ca
part of an application/problem domain.
- CRC cards are created because they are anthropomorphic. They pretend that classes ha
and the analysts and dev team role play an instance of the class being analyzed to eke o
- The needed questions to ask during Object Oriented role play:
    - Who are you, which defines classes
    - What do you know, which defines attributes
    - What can you do, which defines methods
- When a CRC card has a responsibility to fulfill, but not enough information to actually fu
with other classes to get the job done.

CRC card (front side)		
Class Name: Patient	**ID:**	**Class Type** (Abstract, Concrete, Domain): Concrete, Domain
Brief Description: A person that receives/received medical care		**Associated Use Cases:**
UC#	**Responsibilities**	**Collaborators** (classes)
3	Make appointment	Appointment
3	Get last visit	
5	Change status	
2	Provide medical history	Medical history
4		Medical fees

CRC card (back side)	
Attributes	**Relationships** *with other classes*
	Generalization (is a):
	Person
	Aggregation/Composition (has parts):
	Medical history: Composition
	Association:
	Appointment

- A good class model consists of enduring types of domain objects that do not depend on
required today. They are not all encompassing, but robust enough to handle different ki
behavior required by the system must be able to be provided by objects of the classes i
- Data driven design identifies the data in the system and divides it up into classes. Respo
identifies all the responsibilities in the system, and divides them up into classes.
- To find classes using the data driven design, write up a couple of paragraphs to create a
nouns you see in that description, mark them down. Eliminate single instances nouns, r

ine the user requirements

rds represent the whole or

ve human characteristics,
out attributes and methods.

lfill it, it has to collaborate

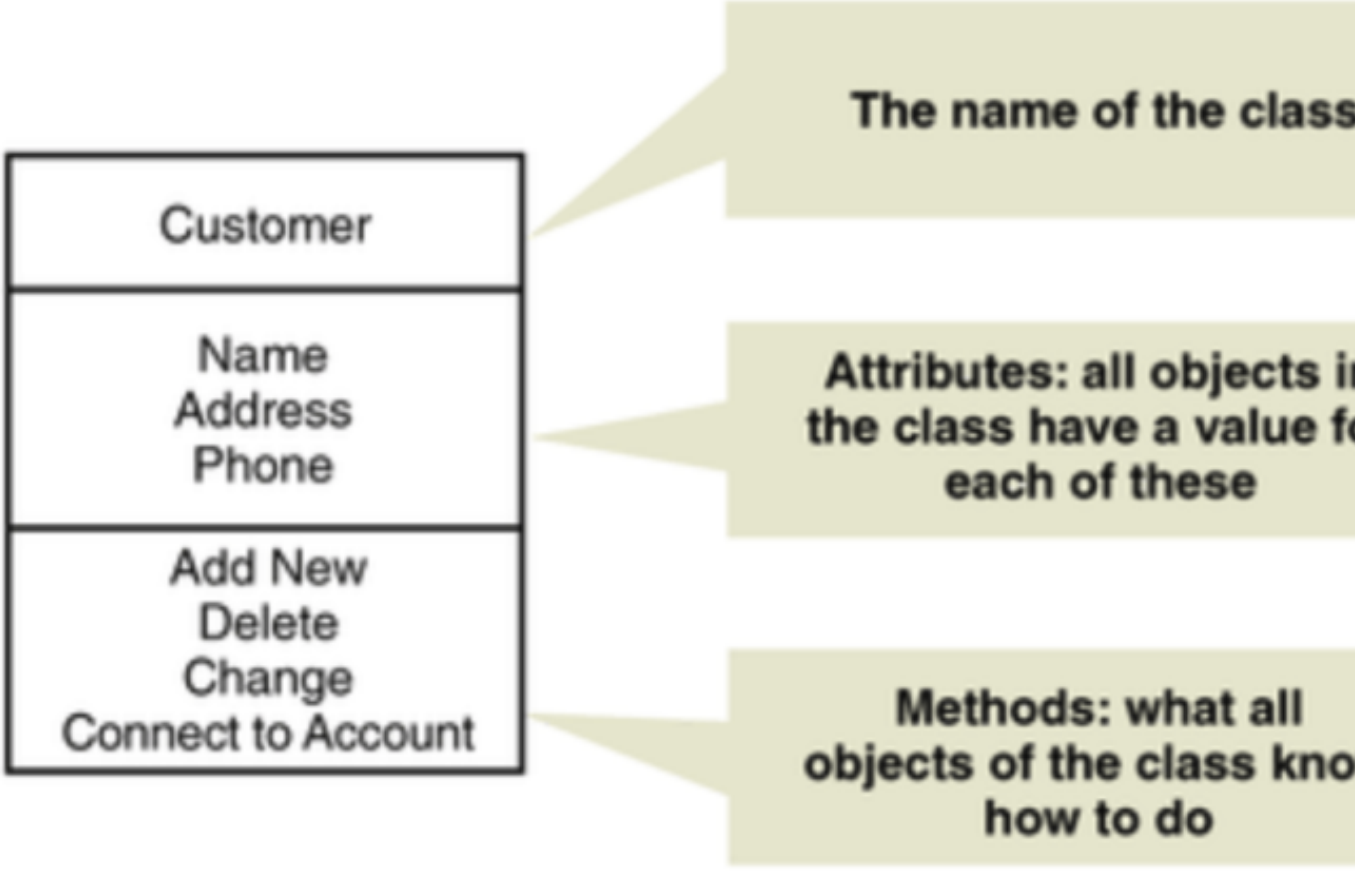
the particular functionality
inds of workflows. All
n the model.
onsibility driven design

system description. Any
edundant nouns, vague

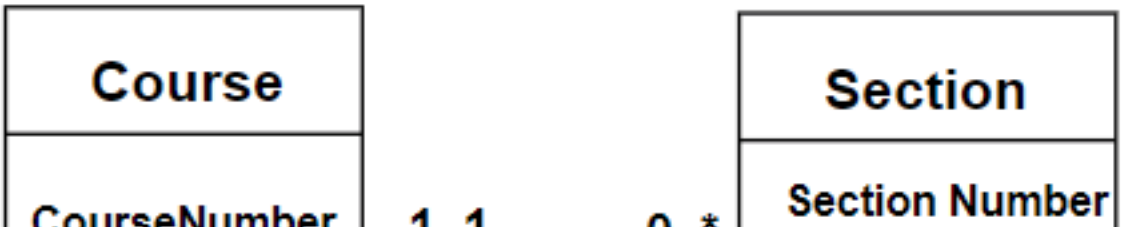
nouns, reports, and events and actions. The pruned list is your new set of classes.

UML Class Diagram

- The class model captures the static features of the system. The class diagram is a proce and a data storage requirements model. The CD captures the class model.
- Attributes can be grouped into classes based on the name of both the attribute and clas your CRC cards.
- Each object in a class should be identifiable. You also need one unique attribute to ID ar person or name for a state. This ID is called an object reference, and is created when th
- The class diagram is an UML model. It shows classes rather than data entities like the ER relationship between classes. It shows how many times the relationship can occur, and methods of the classes as well.
- Below is a class symbol in a UML diagram:



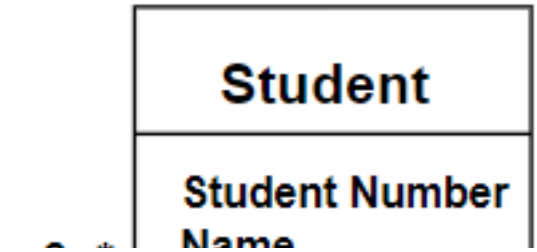
- Below are the relationships in a class model:

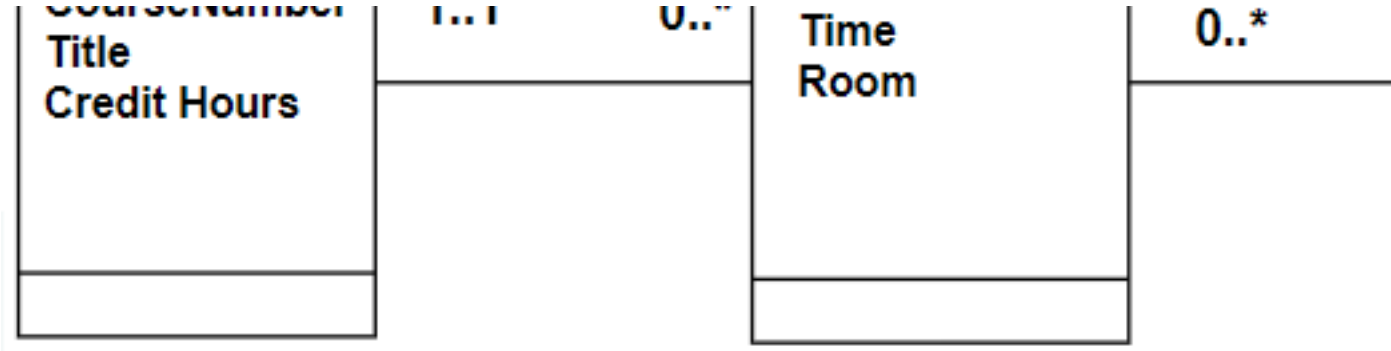


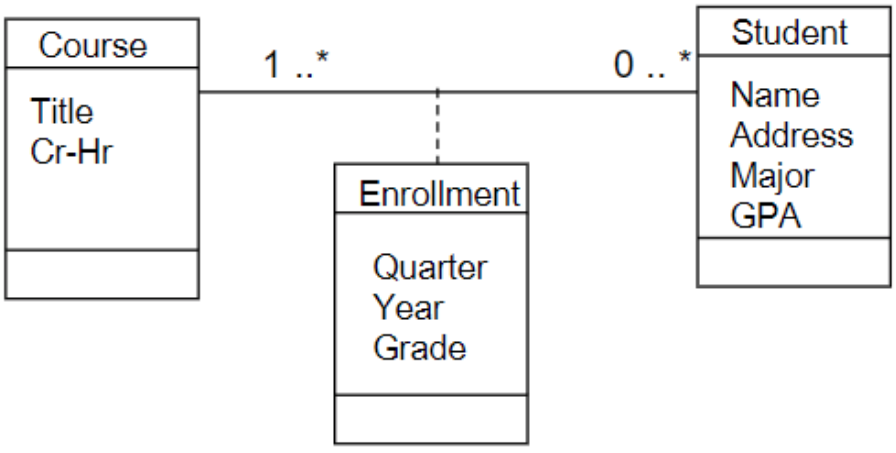
ssing requirements tool,

sses you generated with

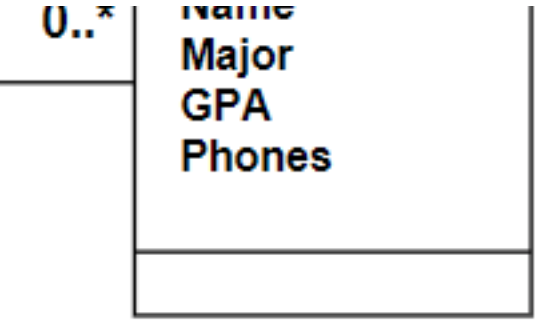
n object, like SSN for a
e object is created.
R diagram, and shows the
can show attributes and



- 

- In OO analysis, the class model is a logical model. It shows what classes of an object are needs to represent these objects to provide the desired functionality. In some instances business or problem domain classes.
- The class model in UML does NOT show how the objects might be implemented, and do might interact with them. No GUI code is here.
- An attribute of a class is one piece of info that needs to be known about the objects in t have different values for the attribute. These values are what an object "knows". These different types.
- An association in UML can be of many types:
  - The optional relationship, where an object CAN be associated with another object
  - The mandatory relationship, where an object MUST be associated with another o
  - The above relationships are described with a multiplicity/cardinality, showing hov participate in the association. In UML, in the example above, the section can only course can have 0 or more sections.
  - Additionally, an association can have a class on itself. This means that the associat has attributes itself. An example of the notation is shown here:
  - 

  - Other relationships exist, like generalization/specialization hierarchies. These have defines inheritance from one superclass to multiple subclasses, where those subc methods of the superclass. Abstract classes are those superclasses that do not ha in the system. Superclasses without this property are just plain classes. A subclass than one superclass.



required, and the system
s, the classes are called

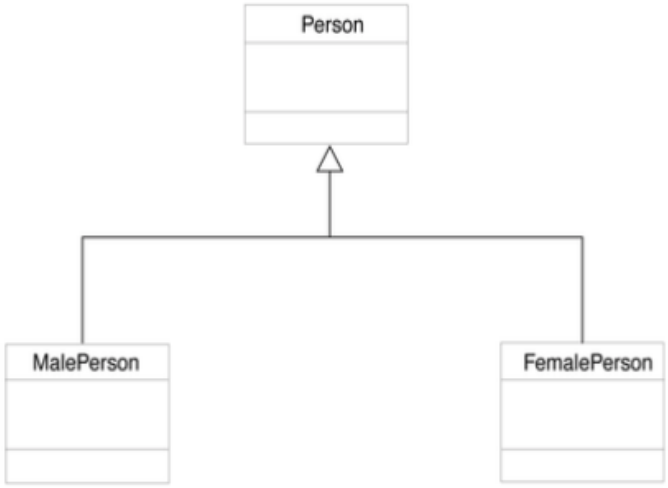
oes not show how the user

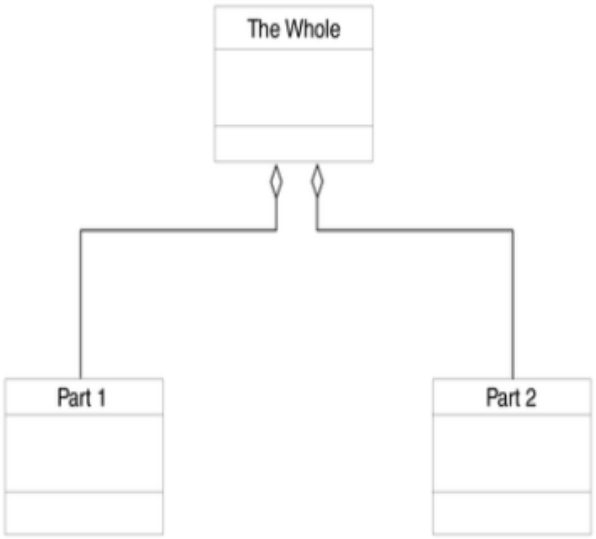
he class. Each object might
values can be of several

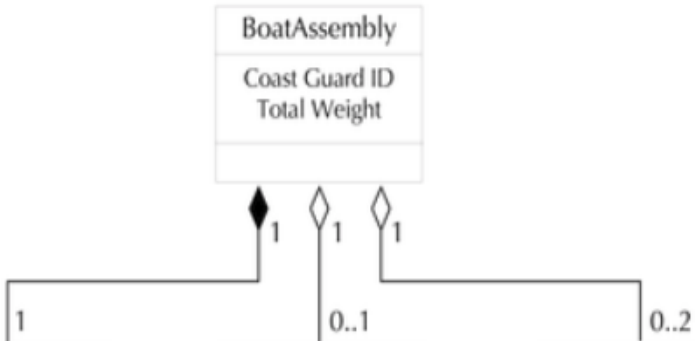
t, but might not be.
bject.
v many classes must
have one course, but a

tion between two classes

e an Is-A relationship. It
lasses share attributes and
ve any instantiated objects
s can inherit from more

- Aggregations are whole-part hierarchies, comprised in a Has-A relationship. This r parts, and defines an object in terms of its parts.
- Gen/Specialization:
- 

- Aggregation:
- 

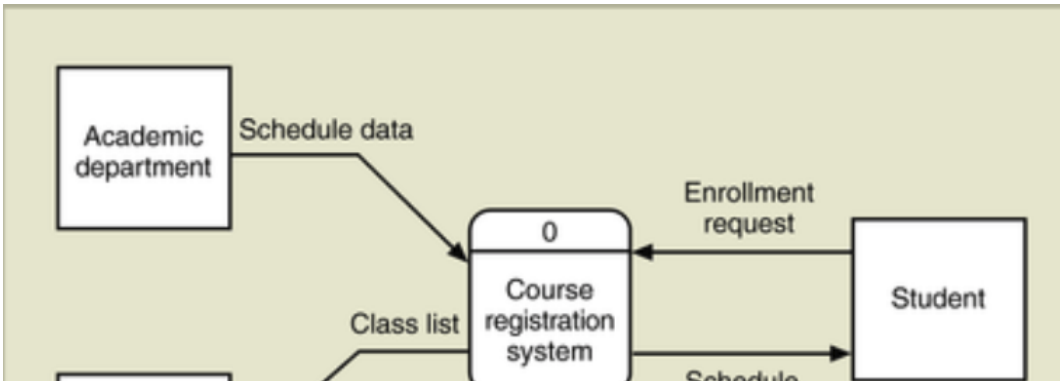
- A special type of aggregation exists called the composition. It indicates when a su more subclasses to exist. The mandatory subclasses are indicated with a black dia one:
- 

elates an object and its

perclass requires one or
mond instead of a white

- Next, we look at the requirement driven approach. To do this, you look at every use cas them against your candidate classes. If you can identify any user story that has responsi belong to an already existing class, create a new one. Otherwise, assign the responsibili class. You can use verbs from your requirements to find relationships between your clas
- Encapsulation is when objects have both their attributes and methods of the class packa means that we restrict access to attributes behind accessor methods, as we "bundle dat way).
- Methods are something the object knows how to do. It represents the behavior of an ol responsibility of an object. Methods come in two types, standard, which are in charge o attributes and instances of a class itself, and custom methods, which apply only to objec class, reflect what part of system behavior the object is responsible for, and are identifie use cases, and asking which objects will be involved in the use case, and what will the ol execute the use case.
- The nomenclature for methods is "verb noun".

Context Diagram

- The context diagram is a picture that shows the scope of the system.
- A context diagram contains:
  - Only one bubble representing the system. The bubble can be a rounded square or
  - External entities/agents representing users of the system that interact with it. Cal These are rectangles or squares.
  - Interactions between the system and its external agents, which contain how the c system.
  - An example is here:
  - 

e or user story and check
bility that should not
ty to an existing candidate
sses.
aged together. (it really
ta with the methods" this

bject, as well as the
f CRUD operations on
cts belonging to a specific
ed by rolling through your
bjects have to do to

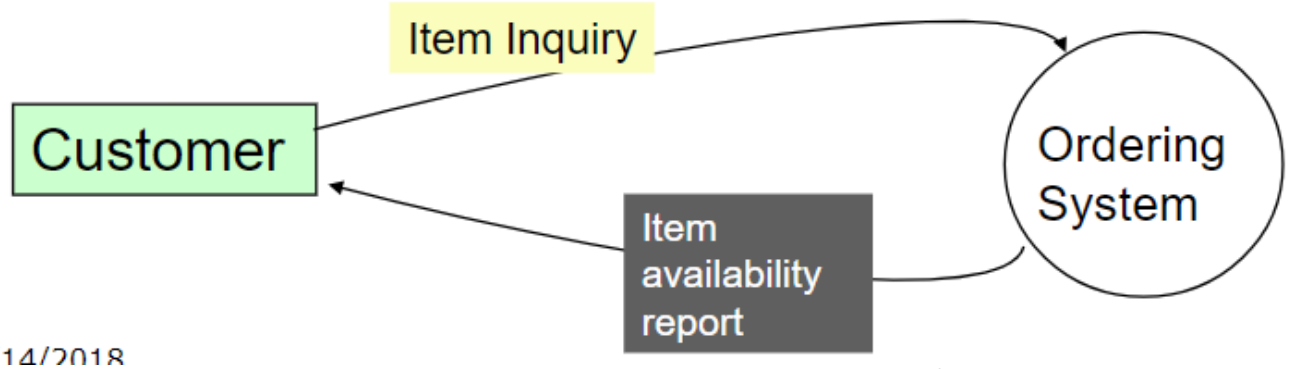
r a pure circle.
led the externals in the CD.

data flows to and from the

- The objective of a context diagram is to capture data going into and out of the system, a data is coming from or going to. It is used to reduce requirements creep later in the pro
- Context diagrams differ from UML, as they feature an expanded scope. It uses a traditio using Data Flow Diagram notation, focuses on data, and has external actors creating or
- External actors are outside the system. They can be people, other systems, or entire gro provide data and get data.
- Data flows indicate interactions between systems and the environment. They represent of the system, not the actions taken to get that data into the system. They are identifie without any verbs at all. The descriptors can be grouped up, like Account Info instead of Blood Type + Fingernail Length
- Inflows to the system are stimulus to an interaction, and outflows are generated as a re These flows are associated with events on the event table, triggered outside the system the system with the response.

Event	Trigger	Source	Activity	Response	Destin
Customer wants to check item availability	Item Inquiry	Customer	Look up Item	Item availability report	Custo

-



14/2018

- As you can see, you can rip a lot of your data flow diagram/context diagram straight fro

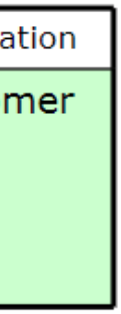
Use Case/ UCD

- SDLC has 2 different approaches, the traditional, where the system is a collection of pro data entities and accept inputs while producing outputs, and the OO approach, where t interacting objects that interact with people and each other that send and respond to m

and to which users that
ject.
nal view of the system,
consuming data.
oups of people. They

t data flowing into and out
d with a noun phrase
f Card + Name + DOB +

sponse to an interaction.
with the inflow and inside



m your event table.

cesses that interact with
he system is a collection of
nessages.

- Many different diagrams are used to document requirements, like the UML and DFDs al
  support each other in areas where their info is lacking.
- When we draw a use case diagram, for each functionality belongs to a type of user.
- The old way of software is that the system is:
  - A collection of processes
  - Processes interact with data entities
  - Uh
- The new way is to use OOP.
- A variety of different diagrams are used to document requirements when using the OO
- UML (Unified Modeling Language) is the standard notation for OO diagrams. It was mad
  Rumbaugh, made by the RSC and OMG.
- In OO, there are diagrams to describe dynamic events:
  - Use Case diagrams
  - Sequence diagrams
  - Collaboration diagrams
- And there are diagrams to describe static things:
  - Class diagrams
  - State charts
- A use case is a single function performed by the system for those who use it. Actors are
  system, and the system is the collection of all of the use cases. The system is essentially
- The use case diagram has circles with the use cases in them, inside a big oval that repres
  system. Then, it has stick figures representing the actors. The actors point arrows at the
  involved in/send data to. Use cases point arrows at actors to identify that they are send
- The use case diagram is a graphical model that summarizes info about actors and use ca
  as a whole. It identifies major uses from the event table, and identifies functions that m
  new system.
- You can also split specific use cases and actors into separate subsystems to make the di
  responsibilities of the system. For example, in the below picture, this could be the item
  subsystem. Another subsystem would be the subsystem that controls the killer drones t
  which itself could have subsystems of its own.
- An example of the UCD is shown here:

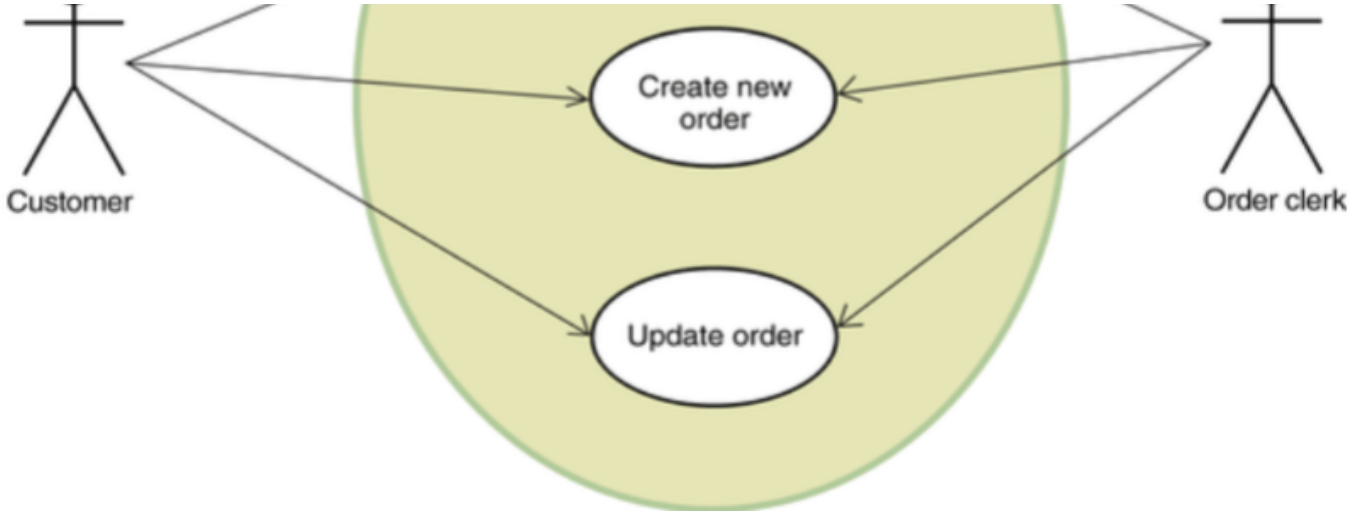
bove. These diagrams

development approach.
le by Jacobson, Booch,

the roles users play in the
the collection of use cases.
sents the internals of the
use cases they are
ing data to the actors.
ases, looking at the system
ust be supported by the

agram cleaner and split up
inventory/logistics
hat deliver the items,

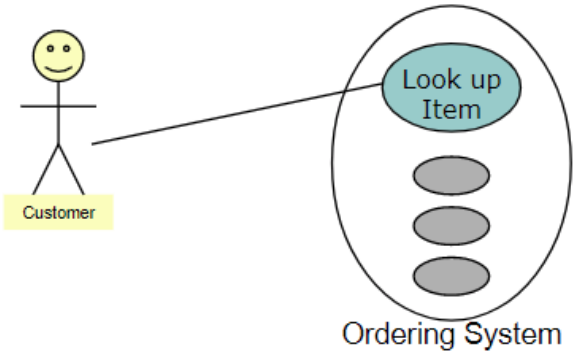
-



- To ID use cases, you can use the event table to identify the system task that will need to to the event. Then, identify the involved actors with that task/event. This means the sou actor, and the activity is the use case.
- When naming use cases:
  - Use verb noun nomenclature.
  - You can use adjectives and adverbs, but you have to have the verb followed by th specific strong verb/noun.
- When naming actors:
  - Use purely a noun or adjective noun phrase.
  - Make sure to reflect roles, not specific people
  - Make sure the actor names are meaningful and specific to a role
- Example of conversion from event table to use case diagram:

Event	Trigger	Source	Activity	Response	Destination
Customer wants to check item availability	Item Inquiry	**Customer**	**Look up Item**	Item availability report	**Customer**

-



- For every use case you identify in the UCD, you have to give a detailed description of th description process, you can ID more use cases that are used by your original use cases

o be performed to respond
urce and destination is your

e noun, with a meaningful,

e use case. During the
from your diagram.

- A **use case** is a sequence of actions a system performs that yields an observable result o actor. It captures the main functionality from an actor's perspective. The sequence of ac referred to as a scenario, and each use case has at least one scenario.
- Each use case has required fields:
    - The **name** of the use case
    - The **summary**, a brief description of what the user is trying to accomplish
    - The **actor**, who is a person or external system outside the scope of the system tha detailed description field
    - The **input** data, a list of all external data needed for the UC to be performed
    - The **output** data, a list of all data returned from use case execution
    - The **basic flow**, which is the happy path that can contain subflows, calculations, lo typically a numbered list of execution steps, and is where the scenario is containe implementation specific language here. Sometimes, you will figure out that a use cases. Each case is distinct and separate from the others, with no relationship bet you will use subflows, one for each distinct case.
    - The **alternative flows**, which is where all exception and error cases are detailed, a scenarios that still involve the use case.
    - The **invariants**, which are a set of conditions that must be maintained throughout put assumptions about operational things here.
    - **Pre-conditions**, things that must be true before the use case flow can begin, and v use case flow completes
    - **Post conditions**, which are conditions that are promised to hold after the scenario
    - **Extension points**, which are where we use <<includes>> and <<extends>> cases. T cases. The **includes** relationship is used when a collection of functionality is used i Common interactions within different use cases can be identified and separated in increases code reuse, and decreases system complexity. The **extends** relationship variation on normal behavior. Extended use cases are only used by the use case th exist without the extending use case. Its used when a subflow or a needed part of complex, and deserves its own scenario due to this fact.
    - **Business rules**, which are the business rationale for the use case, its exceptions, a rules that dictate how the company conducts its business. They can come from wr precedent, and are documented as use cases are described. They are supported b layman's terms, they are facts about the business that explain why the use case sc flows are in place.
    - **Notes**, which is *notes*.

Object Interaction Models

- UML specifies two OI models, the sequence diagram, and the collaboration diagram.
- The sequence diagram presents object interaction arranged in a time sequence. It is use

f value to the particular
ctions required is often

t triggers step one of the

gical structures. This is
d. Do not use
case has several distinct
ween them. This is where

s well as just less common

the use cases. You usually

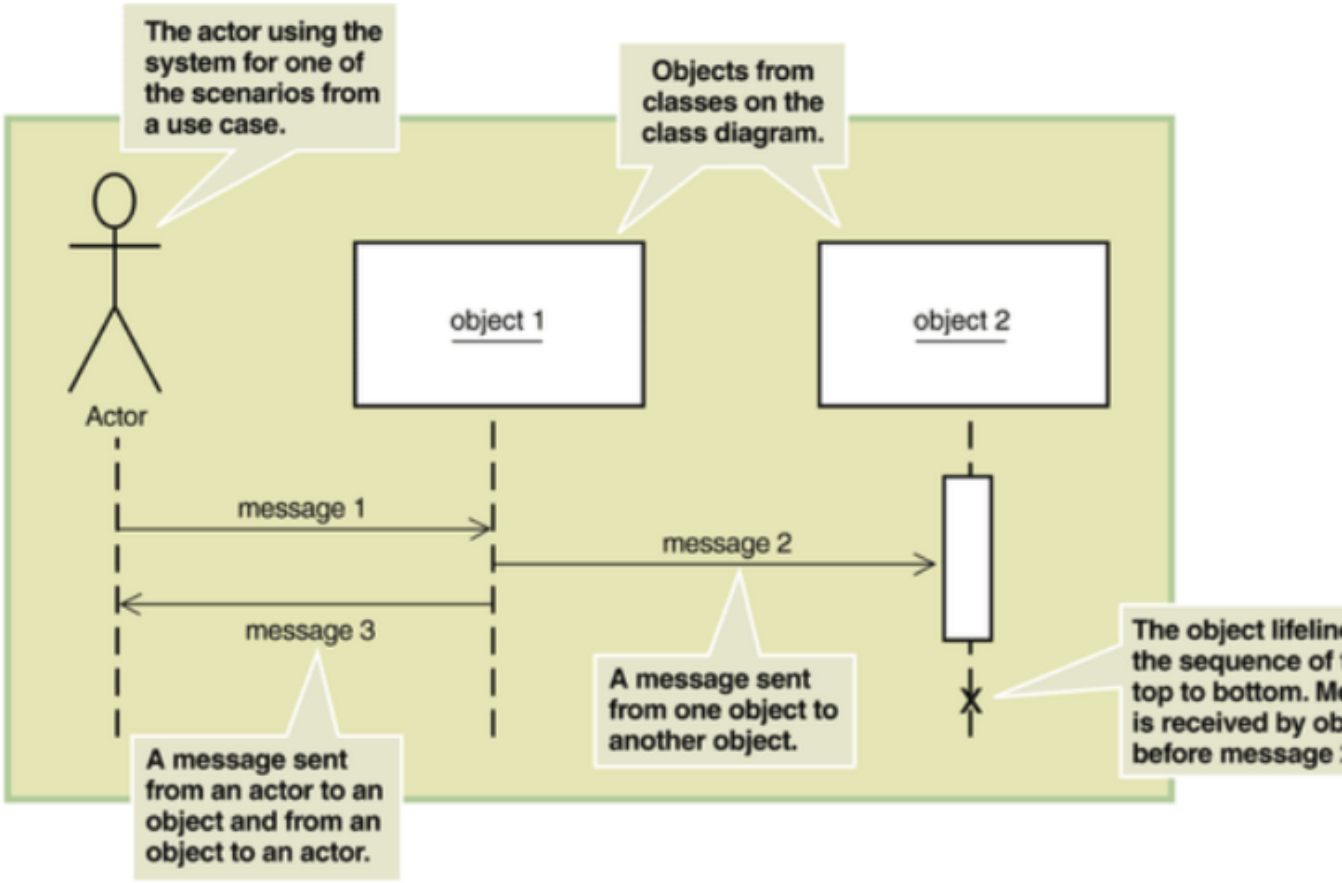
will not be true after the

o completes
These apply to other use
n several other use cases.
nto new use cases. This
is used to describe a
hat extends it. It cannot
a scenario is incredibly

nd its errors. They are
ritten policy or unwritten
y alternate flows. In
cenarios and alternate

d to show the objects

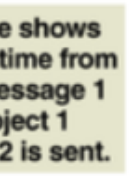
- The sequence diagram presents object interaction arranged in a time sequence. It is use
  involved in a use case scenario, and the sequence of messages that are exchanged.
- A collaboration diagram shows interaction organized around the objects and their mess
  emphasizes the objects that interact together to support a use case diagram.
- Sequence diagrams consist of four basic symbols:
-



- Each object name can take on different forms:
  - Object, the name of the class
  - Object or :object, an unidentified or anonymous Object object.
  - Name:object, an instantiation of Object with Name identifier.
- A vertical line under an object is used to show the passage of time. If it is dashed, the cr
  the object is not important to the scenario being run. Activation lifelines are narrow rec
  show that an object is only active during the time covered by the rectangle.
- The messages are requests from one actor or object to another to do some action, and
  method invocation.
- To develop a good sequence diagram, follow these steps in order:
  - ID all actors involved with the scenario
  - ID which object will get the first message from the actor, and what parameters wi
    actor.

ed to show the objects

sages to each other. It

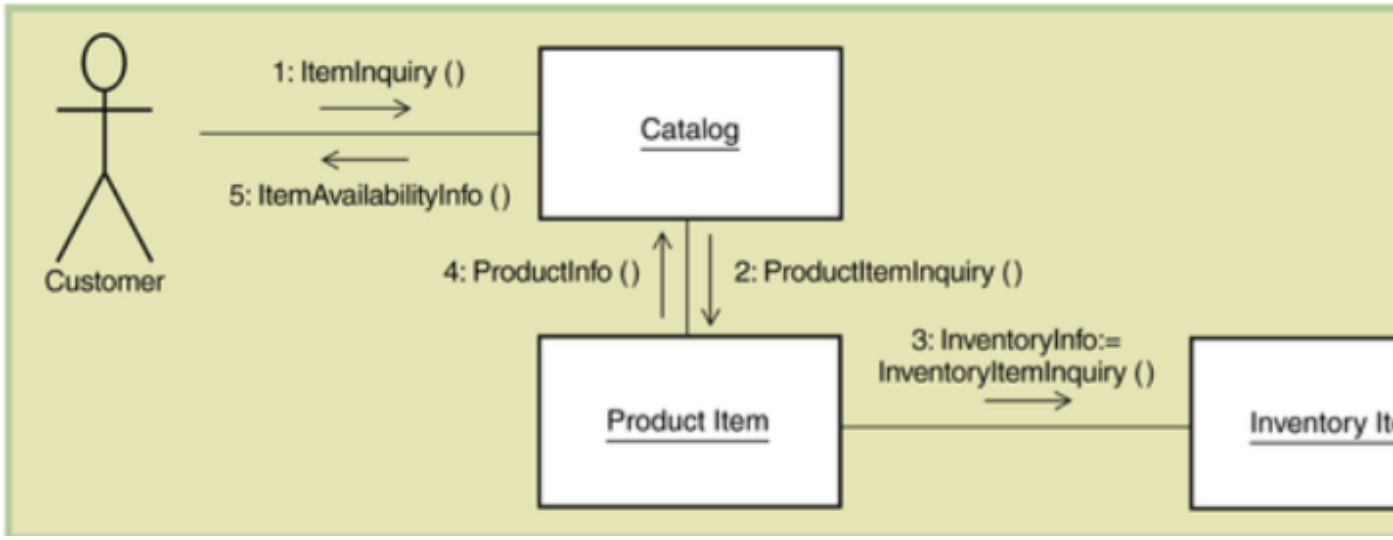


eation and destruction of
ctangles that are used to

these actions are done by

ll be needed from the

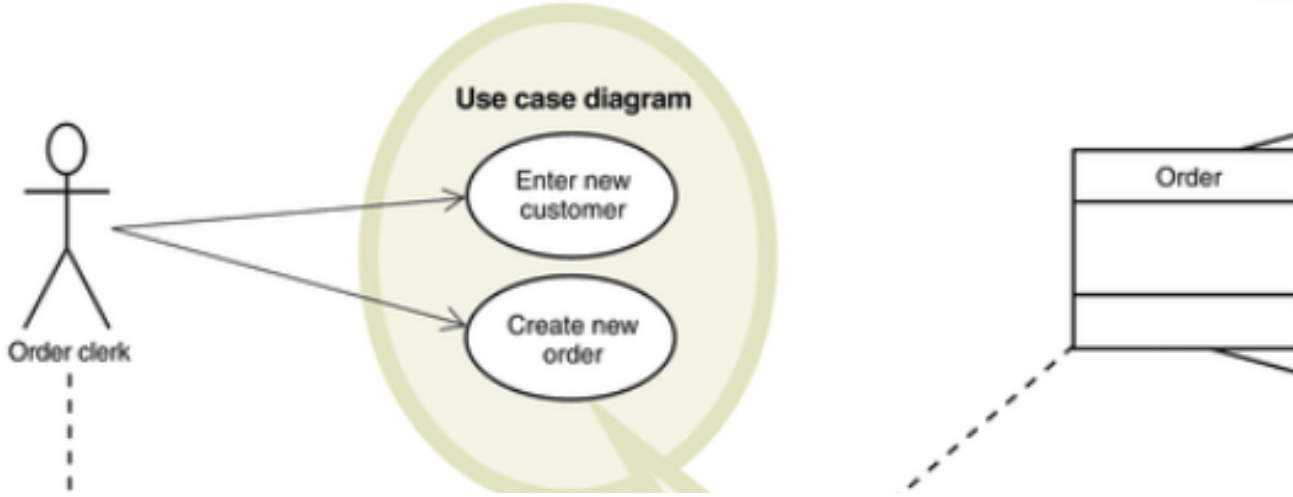
    - ID what objects will have to collaborate to get the use case scenario done.
    - ID what messages the objects will have to send
    - Sequence messages correctly and attach to appropriate lifelines
    - Add response messages and communications to complete the diagram.
    - (If you have an included use case from the <<includes>> relationship, show this as that handles that use case)
- When an object is deleted, put an X on the bottom of its activation lifeline.
- A collaboration diagram is just a sequence diagram without the time information:
-



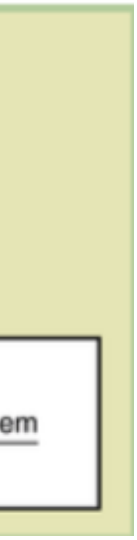
- The sequence diagram uses classes from the UML class diagram to describe a use case c case diagram.

State Charts

- Objects are born, live their lives, and die. Some objects have interesting life histories, ar state transition diagram.
- This is the final piece of info needed for describing functional requirements. It identifies be in, and what messages make the object change its state. The relationship for this nev



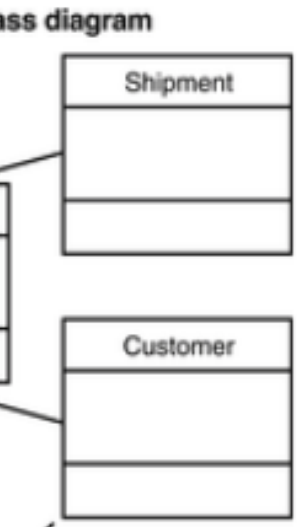
s a message to the object



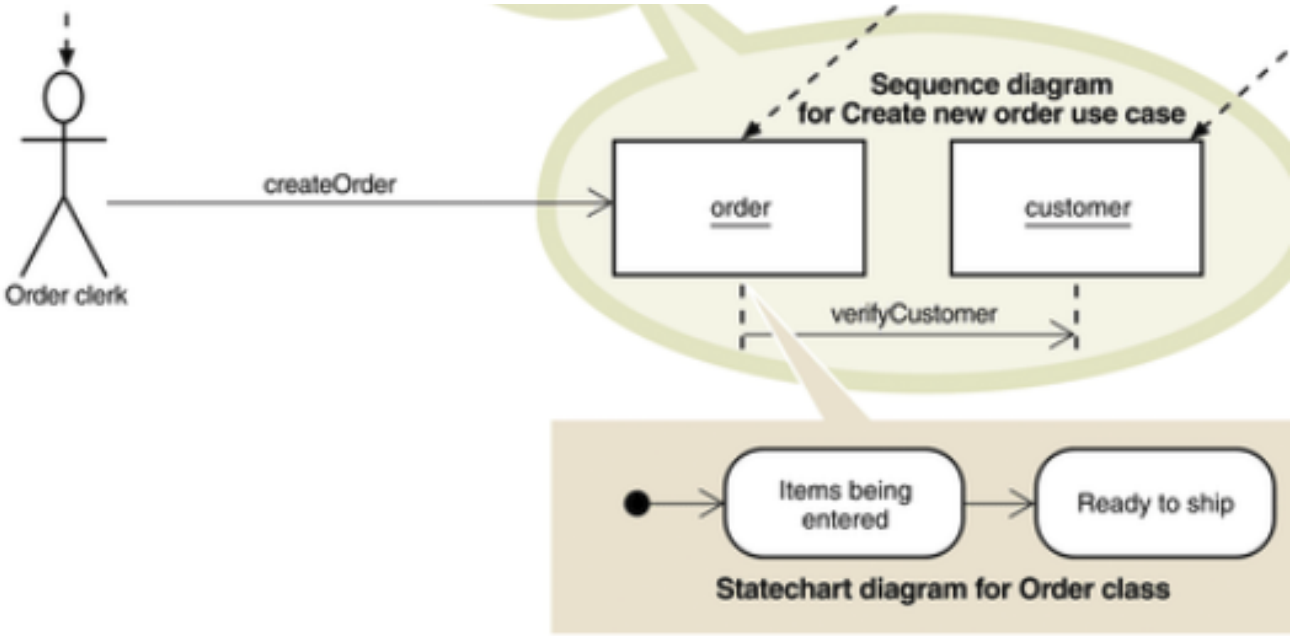
contained within the use

nd can be modeled with a

what states an object can
w diagram is:



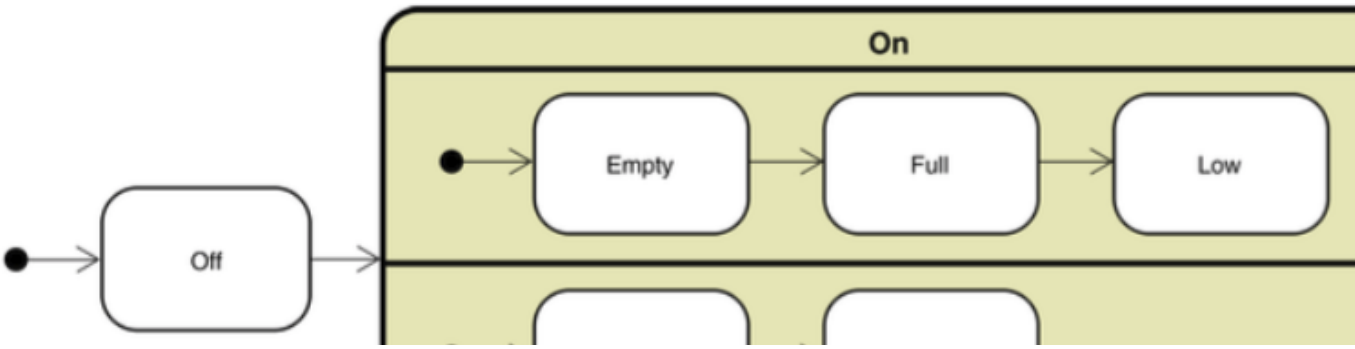
-



- The state transition diagram is only needed for interesting objects. The state transitions within a singular use case, or across multiple use cases.
- A state is a value of some condition that occurs during an object's life. The condition sho An object can only move to a new state due to an external event, or because of an inter from another part of the system is used, or an amount of time passes.
- The naming conventions for object state can be:
  - A simple condition like on, or off
  - Active names consisting of verb phrases
- An object may automatically perform an action when entering or leaving a state.
- A composite state is a state that has other states nested internally. A concurrent object state at a time.

Sample Composite States for a Milling Machine Object

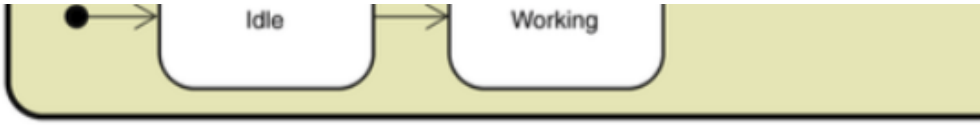
-



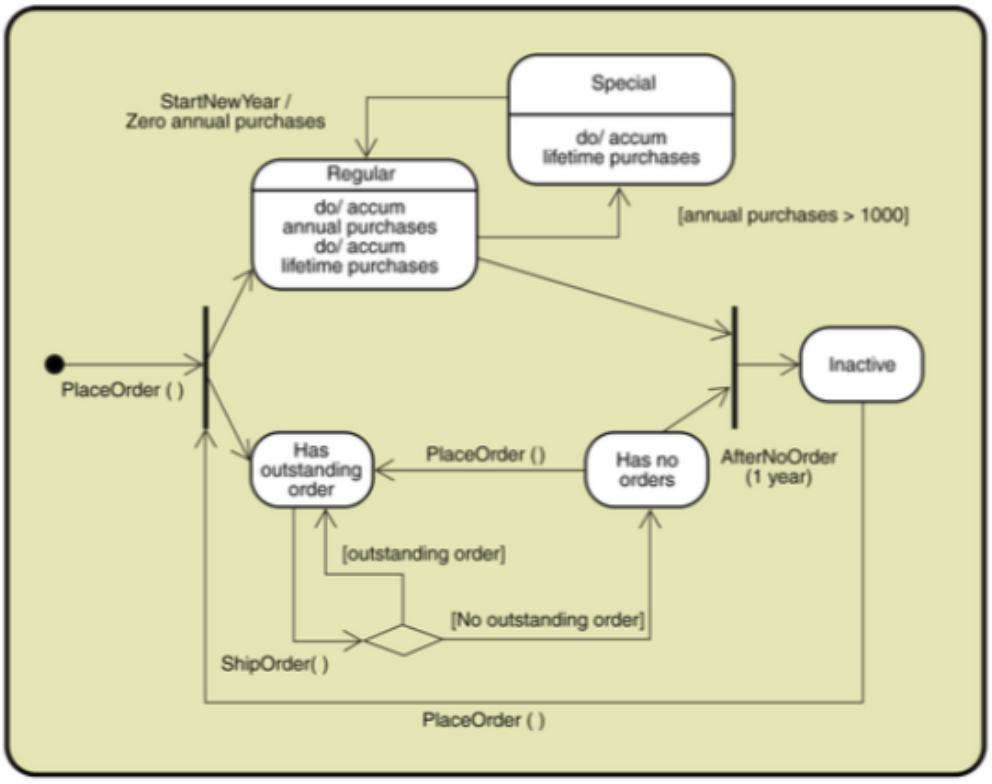
described here can occur

ould be semi-permanent.
rnal change, like when data

can be in more than one



- An object transition is an arrow that can be labeled with 3 pieces of info:
  - The message that causes an object to leave one state and change to a new state
  - A [guard condition] that must be true for the state to change
  - An action-expression := that happens when the transition occurs.
- Actions can also be shown in the state. These have specific labels for when the action oc
  - Entry is when the state is entered
  - Do is the action executed while in the state
  - Exit is the action executed before leaving the state
- To develop a state chart, look at all the sequence diagrams that involve the object, list t output messages, and develop the states and transitions based off of these two things.
- A composite state can show an object doing things concurrently. Refer to the previous f this. Another way to notate this is to use vertical bars to indicate when concurrent beha are called pseudostates:
-



ccurs:

he relevant input and

igure on how to notate
avior begins and ends that