

Agile Requirements Visualization

**Christian Sioui
March 2018**

At the end of this presentation, you should:

1. Be familiar with requirements modelling within an agile team
2. Have the fundamentals for a Domain Model
3. Have enough knowledge to seek advise on how to visualize other requirements

Classes and Objects

- Problem Domain Entities
 - Business Entities involved when actors are using the system – when performing all the use cases
 - Examples:
 - Claims
 - Providers
 - Members
 - Insured Dependents
 - Vehicles
 - Pre-Certifications
 - Inquiries
 - Recall letters

Class versus Object

- A CLASS is a collection of objects that share similar qualities, such as:
 - Attributes
 - Operations
 - Relationships
 - Semantics
- AN OBJECT, on the other hand, is an instance of a class
- Example:
 - The Provider class represents the collection of all the Providers
 - St. Mary's Hospital provider is an object belonging to the Provider class
 - When actors want to create a new Provider, they create a new object, which is an instance of the Provider class

Finding Entities

- A simple technique for discovering classes is known as *Grammatical Analysis*
- Identify the nouns in the Problem Statement
- Circle every noun (noun-verb pair / adjective-noun pair)
- Of these nouns:
 - Some will become Classes
 - Some will become Actors
 - Some will become Attributes of a class
 - Some will have no significance at all on the requirements
 - Refer to the Acid Test / Challenge Questions for each noun

Discovering Classes

- Classes can be discovered from different sources:
 - Domain Knowledge
 - SME
 - Glossary
 - Previous systems
 - Similar systems
 - Enterprise models
 - Reference architecture
 - Vision Document
 - Problem Statement

Acid Test / Challenge Questions

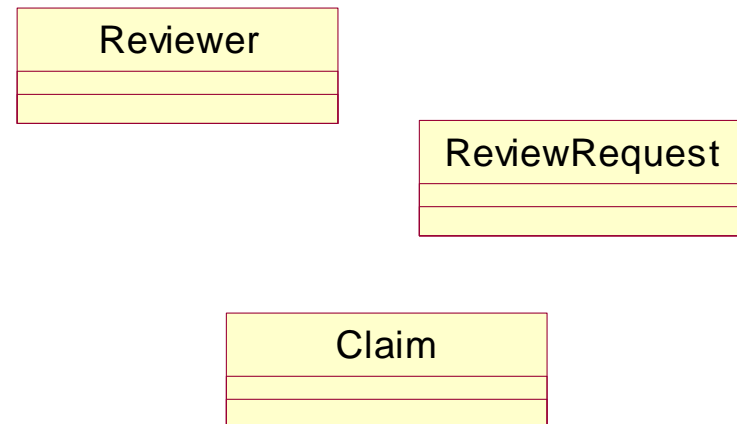
1. Is this candidate inside our system boundary?
If not, it might be an actor
2. Does this candidate have identifiable behavior?
Can we name the Services / Responsibilities this candidate provides?
3. Does the candidate have a Structure?
Can we identify some information this candidate owns?
4. Does this candidate have relationships with other candidates?

Problem Statement

We need a new website that will help capture the scores of your junior basketball league. There are 8 teams and more than 100 players and we can't keep updating each player by phone or print a *Team Sheet* anymore. As soon as the website becomes available, we will tell players (and their family) to go to the website to go see the scores online, to view the schedules, the playoffs, or last minute cancellation due to snow storms.

Class Name

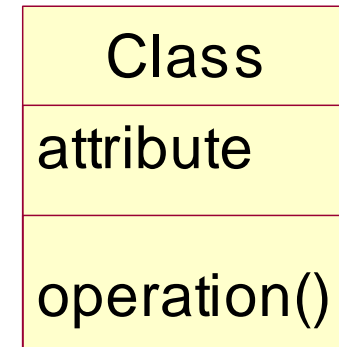
- No Spaces
- No Special Characters
- Title Case
 - First letter of each noun is capitalized; the rest, lowercase
- Singular form (as opposed to plural)



Notation & Responsibility

Notation

- Rectangle
- 3 compartments:
 - TOP compartment contains the Class Name
 - MIDDLE compartment contains the Attributes
 - BOTTOM compartment contains the Operations



Responsibility

- Describe the responsibility of the class
- What is the class responsible to accomplish?
- What part does it play to provide the required system functionality?
- For example: the Reviewer class is responsible to provide an opinion (the answer to a Review Request); and maintain its information.

Describe the Class Responsibilities

- For each analysis class
- Describe the responsibility of the class
 - That's the Mission Statement
- Describe the services that the class will provide
 - And that no other class will provide
 - Responsibilities in different classes must not overlap
- Give a class *Crisp Boundaries* / clear definition of what it does (and does not do)
- Adjust responsibilities as we learn more
 - That is refactoring

- Caution: Attributes are not the same as table fields
- Characteristics (or qualities) of Entities
- Attribute Name
 - No space
 - No Special Characters
 - Title Case, except the first word
- Examples:
 - memberLastName
 - providerAddress
 - isDeleted
 - reviewed

Derived Attributes

- Class characteristic derived from the information found in other attributes
- For example, age based on the date of birth
- Although redundant, derived attributes are helpful at design time
- Put a slash in front of a derived attribute

Claim
claimID
claimType
/isPaid
paidDate
paidAmount
serviceDate

Example

When “paidDate” and “paidAmount” attributes contain a value, then the “isPaid” flag must be yes

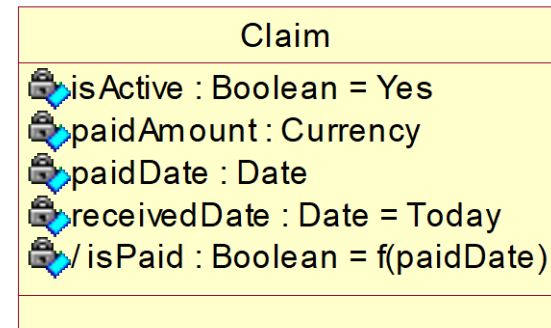
Attribute Type

- A ***constraint*** imposed on the attribute so that it can only hold values as defined by the type
- Example, Boolean, Date, Real and String
- A colon must separate the attribute name from its type

Claim
claimID
claimType : ClaimType
isPaid : Boolean
paidAmount : Currency
paidDate
serviceDate

Initial Value

- Indicates the value given to the attribute when the object is created
- Include an equal sign '=' before the initial value



- Characterize the services provided by a class
- Either:
 - The methods for accessing and modifying the class' attributes
 - Implement the class' behavior
- Operations basically allow the class to carry out its responsibilities
- A Method is an instance of an Operation – a class owns operations; an object owns methods










Operations - Notation

- Specific syntax to provide details about their type, arguments, and visibility
- Arguments are found inside the brackets next to the operation's name
- When a return type is provided, a colon must precede it

Example

The checkEligibility operation:

- Member is an argument
- Return type is Boolean

Claim	
	claimID : Integer
	claimType : ClaimType
	serviceDate : Date
	isPaid : Boolean = f(paidDate)
	paidDate : Date
	paidAmount : Currency
	getPrice(provider) : Currency
	requestPayment()
	checkEligibility(member) : Boolean

- Objects are instances of a class
 - Same squared rectangle
 - Their name is slightly different

 - Object name is underlined, and separated from its class name with a colon
- Emille Garcia : Reviewer
- Anonymous object
 - Omit the name of the object
 - Keep the colon
 - Keep the underline
- : Claim

Relationships

How to Relate Classes Together

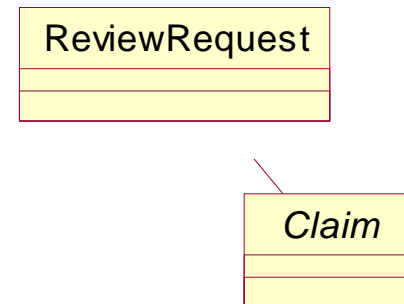
Relationship Types

- Objects need to communicate amongst each other
- Provides the behavior
- Collaboration to accomplish the functionality

- Add Relationships between Classes
 - Association
 - Aggregation
 - Dependency
 - Generalization

Association

- An Association is simply an indication that a communication (or collaboration) exists between two classes
- Does NOT explain the details of the implementation
- Shows that objects communicate
- Solid lines between classes



Navigability

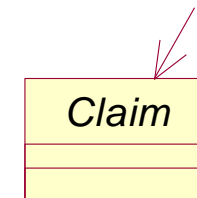
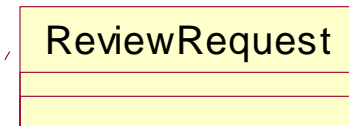
- Navigability refers to the class or object that knows about the other class (or responsible for the collaboration)

Note: At the analysis stage we do not worry too much about how the collaboration is performed, since the mechanism chosen for implementing this knowledge will be specified at design

- An arrowhead points to the class that is "known about" by the other class

- Example

- Claim has no knowledge that it is chosen for a review
- Request must know what must be reviewed
- The navigability is from ReviewRequest to Claim



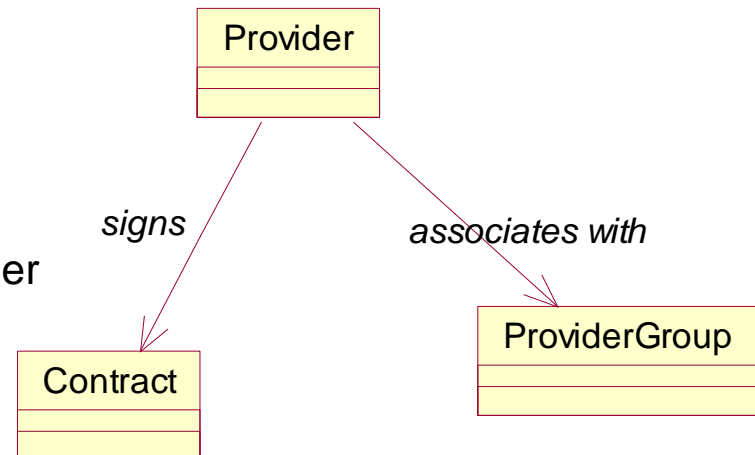
- Relationships with no navigability indicates that both classes know about the other class
- By default, bidirectional, unless specified
- Data can flow in either direction across a bidirectional association

Association Name

- Associations can be named
- Generally an active verb that conveys the meaning of the relationship
- Verb phrase usually implies a reading direction
 - Should read correctly from left to right / top to bottom
- Look at the association in two different directions
- Good practice to provide meaningful names

- **Example**

Provider class signs a contract in order to become a BCBSF Network Provider the contract is signed by the Provider

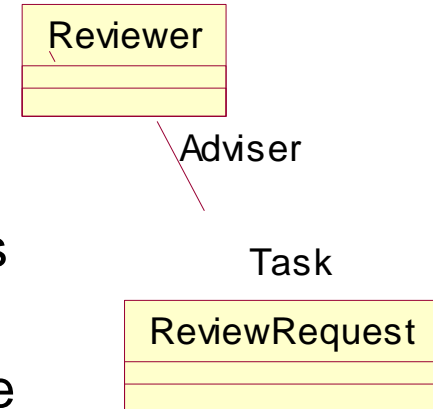


Association Roles

- Generally a noun that conveys how one class relates to the other
- Associations have two Roles
- Role names are better at conveying the meaning of the relationship
 - Bidirectional relationships
 - Difficult to find a phrase that reads correctly in both directions
 - Improve readability
- Do NOT use both

Example

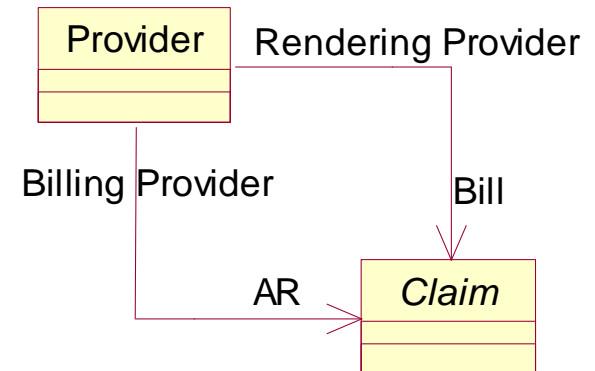
- Reviewer in relation to a Review Request plays the role of an "Adviser",
- Review Request plays the role of a "task" to the Reviewer – it is a task the Reviewer must do (to provide an opinion)



Multiple Associations

Example

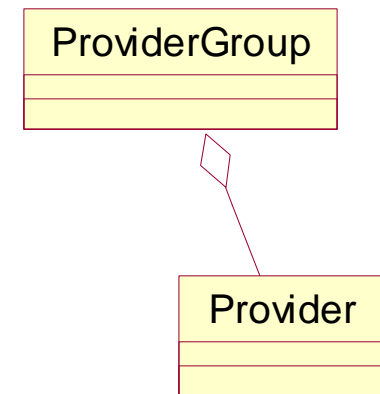
- A Provider can play two roles in relation to a Claim. It can be a Billing Entity or the Provider who rendered the Health Care Service



- Possible but caution!
- Multiple associations between a pair of classes
- Always name all associations to distinguish them
- Note: Although a legitimate construct; do not accept readily multiple associations; challenge them to determine if they reveal separate semantic relationships and not solely individualized behavior of the same relationship

Aggregation

- An aggregate object is one that is made up of components
- Special type of association that models a “whole-part” relationship between an aggregate (the whole) and its parts
- Also known as a ‘part of’ or ‘containment’ relationship
- Notation - Diamond next to the aggregate class
 - Note: The diamond can be attached to only one end because an aggregation relationship is intrinsically one-way
- Aggregations can have:
 - a name
 - can be navigable
 - can have role names



Example

- A Provider can be ‘part of’ a Provider Group.
- The Provider Group is the whole (the aggregate) and the individual Providers are the Parts

Aggregation Types

- There are two Types of Aggregates:
 - Shared Aggregates
 - Composition Aggregates

Shared Aggregates:

- Parts in a shared aggregate can be parts of other aggregates
- Example: a Provider Group is composed of Providers, and one Provider can be part of many Provider Groups
- For Shared Aggregates -- Destroying the aggregate does not necessarily destroy the parts
- Although a Provider Group might be disbanded, its members still exist and are free to join other Provider Groups

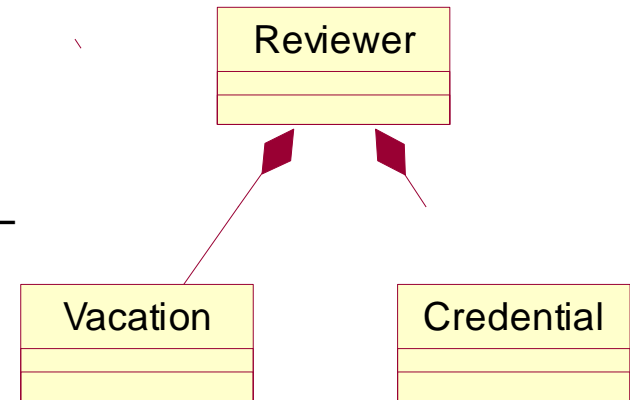
Composition Aggregate

Composition Aggregates

- Aggregate 'is composed of' the parts
- Has strong ownership of its parts
 - Parts will be destroyed if the whole is destroyed
 - Parts cannot be part of other aggregates
- Notation – filled diamond instead of a hollow one

Example

- Reviewer has Vacations and Credentials
- Vacations belongs to only one Reviewer – cannot be shared
- If you destroy the Reviewer, you also destroy the Vacation and the Credential objects associated with the Reviewer

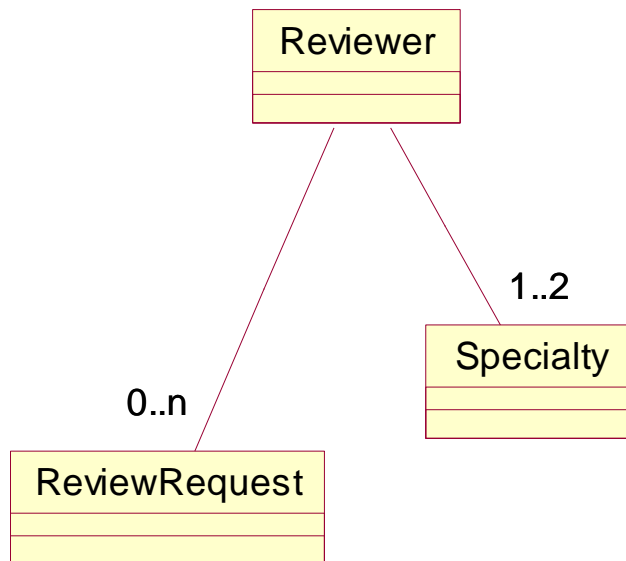


Model the relationship as an aggregation:

- If two objects are tightly bound by a whole-part relationship
- If some operations on the whole automatically apply to some of its parts
- If there is an intrinsic asymmetry to the relationship where one class is subordinate to the other
- When two objects are normally considered as independent, even though they may often be linked, the relationship is only an association

Multiplicity

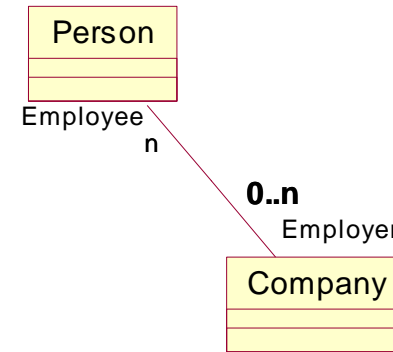
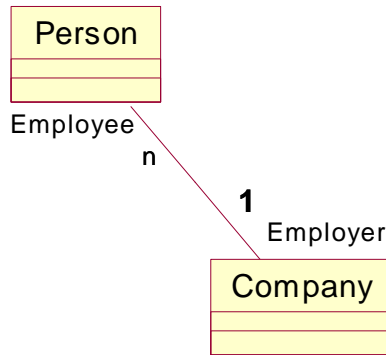
- The number of objects that can take part in a relationship
- Refers to objects, Specified for classes
- Indicated with numeric symbols called multiplicity indicators
- Two multiplicity indicators for every binary relationship



- Multiplicity indicator can be:
 - A single integer
 - A set of disconnected integers
 - A range of integers
 - An infinite number(n)

Making Multiplicity Decisions

- Consider the “Works-For” relationship (or the Employee – Employer relationship) between the Person class and the Company class
- Should the association be one-to-one, one-to-many or many-to-one, or many-to-many?



Relationship Refinement

Refine relationships by including the following four features:

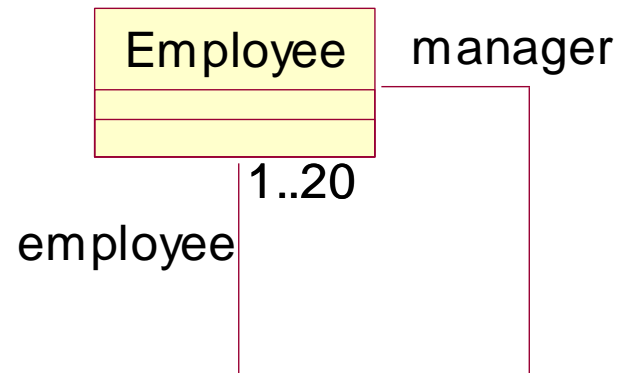
- Reflexive Relationships
- Qualified Associations
- Constraints
- Association Classes

Reflexive Relationships

- Different objects belonging to the same class sending messages to each other
- Shown as a reflexive association
- Role names are used for reflexive relationships (never association names)
- Also apply to Aggregations

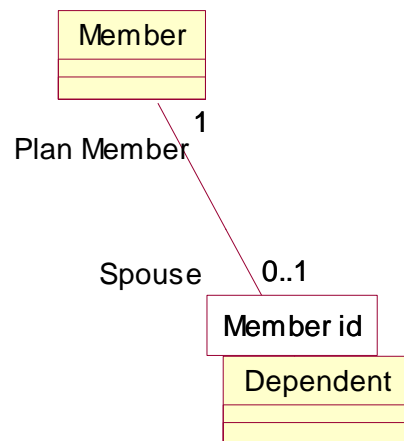
Example

- Let's say that in the organization, a manager can manage up to 20 employees, no more
- A Manager happens to be an Employee as well



Qualified Associations

- Qualified associations are used with one-to-many and many-to-many associations
- The qualifier distinguishes among the set of objects at the “many” end of an association, so to make it unique
- It specifies how a particular object at the "many" end of the association is identified
- May be looked as a kind of key to separating all the objects in the association
- The qualifier is shown as a box at the end of the association



Example

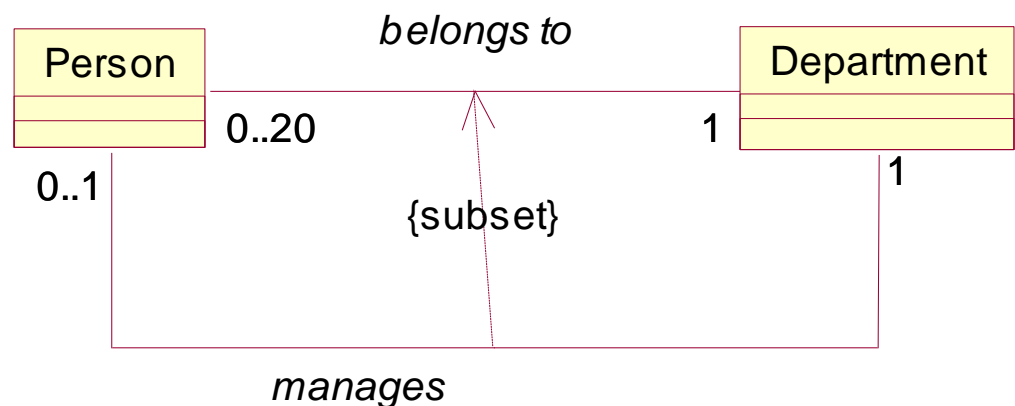
- Amongst the Dependents, a Plan Member has 0 or 1 Spouse

Constraints

- A constraint either limits the applicability of an association or gives more detail to its meaning
- Shown as a dashed line between the associations involved in the constraint
- Specify details of the constraint with comments in braces

Example

- Let's say there are up to 20 employees who can belong to a department; but only one Manager (also an employee) manages the department. Therefore, the employee who 'manages' the department must also 'belong to' the department.



Association Class

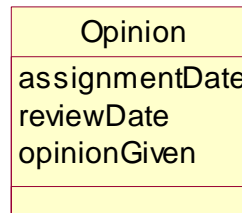
- Relationships sometimes have structure and behavior of their own
- When there is information describing the relationship as opposed to the objects
- The structure (and behavior) belonging to the relationship is modeled as an Association Class
- May include several properties describing the association, but only one association class per relationship
- Frequently used for many-to-many associations

Association Class - Example



Relationship between
Reviewer and **Review Request** is one-to-many

- A Reviewer is assigned zero, one or several Review Requests



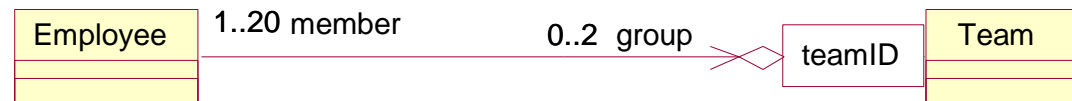
Where to place the Assignment Date?

- The Assignment Date is not a characteristic of the Review Request
- Nor does it describe the Reviewer
- It describes something about when the request got linked to a specific reviewer – it describes the relationship
- An association class holds the link information

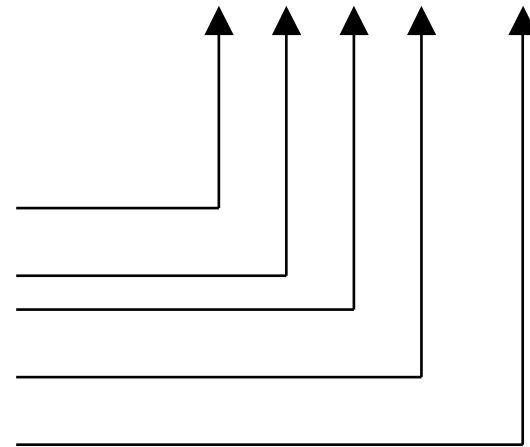
Putting it all together

- The following sequence order should be respected when adding Adornments to relationships:

1. Multiplicity
2. Role name
3. Navigability
4. Aggregation
5. Qualifier



1. multiplicity
2. role name
3. navigability
4. aggregation
5. qualifier



Example

- An employee can belong to no more than 2 teams, and teams can comprise up to 20 Team Members

Q / A