ID:

Object-Oriented Modeling (OOM)

- Abstract description of system structure/behavior
- Language independent
- Object diagram is an instant of class diagram

Visibility	Symbol	Accessible
Public	+	all
Protected	#	self, subclass
Private	-	self
Package	~	same package

Specialization(subclassing) [is-a relation]

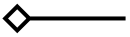
- When certain attributes or operations only apply to some objects of a class, a subclass can be created
- Coded subclass1 extends -> subclass2 extends subclass1

Generalization (bottom-up process)

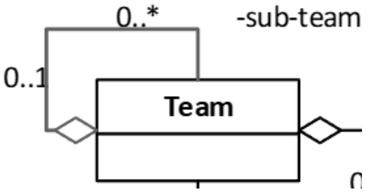
- Process of extracting shared characteristics from ≥ 2 classes and combining into a generalized superclass
- Coded subclass extends

Association

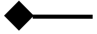
- Bi-directional
- Treated as a second-class attribute of the end class, and is coded as an instance variable

Aggregation [has-a relation] 

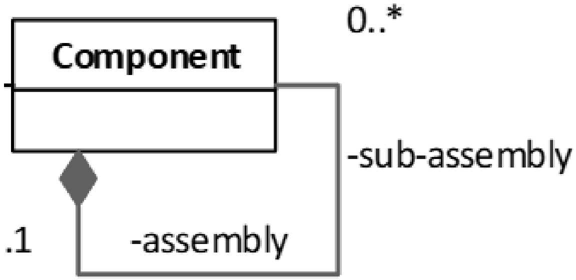
- Whole-part relation
- Classes won't be destroyed if remove one another
- E.g.: A person (whole) has a chair (part), when remove the whole the part still exists
- Coded add



Shape always points at whole not part

Composition [has-a relation] 

- Stronger form of aggregation
- Whole and parts have coincidence lifetimes. When remove the whole part will not be exist
- E.g.: Human (whole) and nerve system (part)
- Coded create -> add



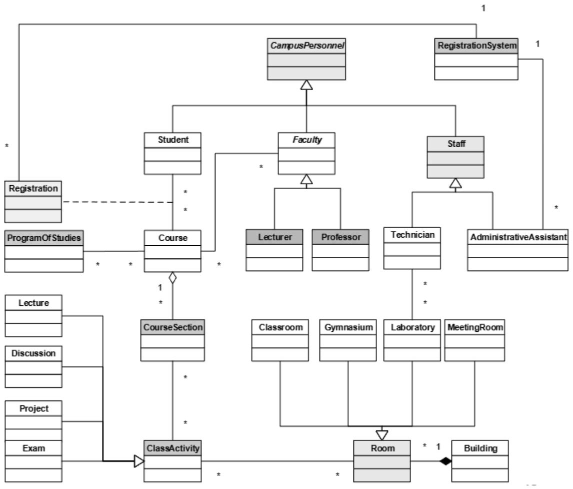
Association multiplicity

Set

- One-to-many associations (X 1→* Y), each X object may linked with more than one Y objects
- Ordered set: collection without duplication. Possible to refer to elements by index, e.g. 1st edge, last edge
- Bag: collection with possible duplication
- Sequence: ordered collection with possible duplication

Dependency ------>

- Not an attribute of part class, used in the form of a temporary variable in code. Only exist during the method call
- Coded implements -> override



Requirements engineering: Process of establishing services that a business requires from a system and the constraints under which the system operates and is developed

Domain knowledge: knowledge of the environment in which the target system operates. May include user workflows, data pipelines, business policies, configurations and constraints

Domain analysis: the activity to understand the business domain that is pertinent to the problem at hand

Domain specialist/expert: people who possess the domain knowledge

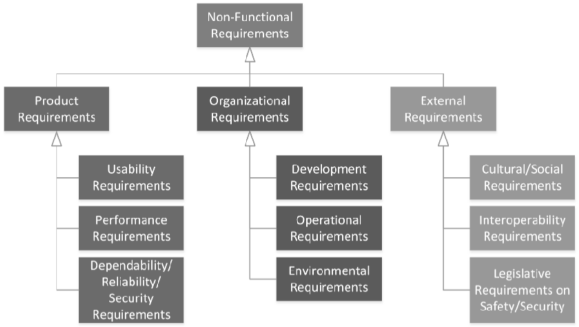
User requirements elicitation

- Difficult due to ambiguous(The MentalCare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month: cost only or by categories?), incomplete, inconsistent, amalgamate (the system shall generate monthly management reports in 5 seconds: functional or non-func?), non-testable (easy to use: same for all?)
- Different user groups might raise conflicts of interest

Functional requirements (services): Statements of services the system should provide.

Non-functional requirements (quality of services):

- constraints on services offered by the system. E.g. response time
- constraints on development process, standards. E.g. UML, web service standards
- constraints on external factors. E.g. laws


Non-Functional Requirements
Product Requirements
Organizational Requirements
External Requirements
Usability Requirements
Performance Requirements
Dependability/ Reliability/ Security Requirements
Development Requirements
Operational Requirements
Environmental Requirements
Cultural/Social Requirements
Interoperability Requirements
Legislative Requirements on Safety/Security

- Often affect the overall system rather than some individual components (may have to design whole new systems with minimize communication between components, unauthorized user redirected to login page whenever a protected resource is accessed)
- Maybe more critical than functional requirements. A system with 5/15 services fault still can provide 10 services. In critical system, non-func requirement not met could cause the whole system to be useless/cause great damage (anti-missile system)

User stories

As a <role>, I want <feature> so that <benefit>

As an administrator, I want to approve photos before they are posted, so that I can make sure they are appropriate.

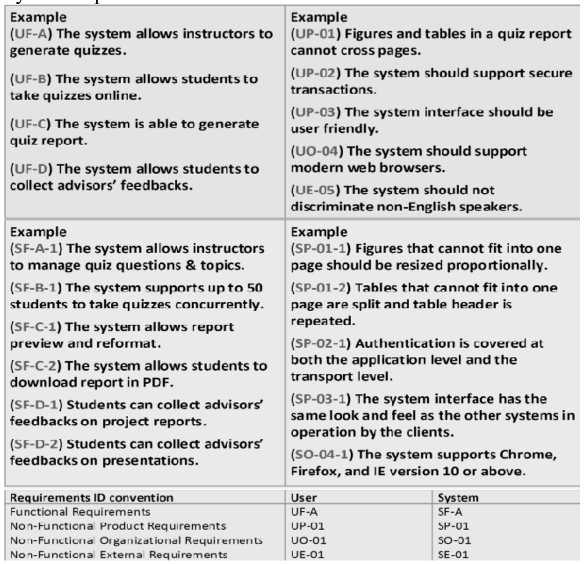
As a sales associate, I want to be able to pull up my current active leads, deals and tasks on my iPhone, so that I can still follow up with clients and update deals status while traveling.

Given < >, When < >, Then < >

Given that I'm a sales associate and I have 30 deals, when I open the App on my iPhone, then I should see the first ten deals, and I should see pagination for the remaining 20 results.

Given that I'm a sales associate and I have zero leads, when I open the App on my iPhone, then I should be shown a button to "Add a Deal."

System requirements Elicitation

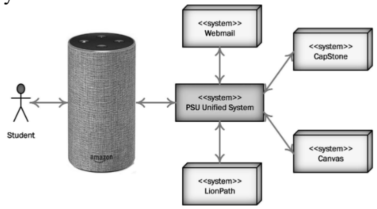


Example (UF-A) The system allows instructors to generate quizzes. (UF-B) The system allows students to take quizzes online. (UF-C) The system is able to generate quiz report. (UF-D) The system allows students to collect advisors' feedbacks.	Example (UP-01) Figures and tables in a quiz report cannot cross pages. (UP-02) The system should support secure transactions. (UP-03) The system interface should be user friendly. (UO-04) The system should support modern web browsers. (UE-05) The system should not discriminate non-English speakers.
Example (SF-A-1) The system allows instructors to manage quiz questions & topics. (SF-B-1) The system supports up to 50 students to take quizzes concurrently. (SF-C-1) The system allows report preview and reformat. (SF-C-2) The system allows students to download report in PDF. (SF-D-1) Students can collect advisors' feedbacks on project reports. (SF-D-2) Students can collect advisors' feedbacks on presentations.	Example (SP-01-1) Figures that cannot fit into one page should be resized proportionally. (SP-01-2) Tables that cannot fit into one page are split and table header is repeated. (SP-02-1) Authentication is covered at both the application level and the transport level. (SP-03-1) The system interface has the same look and feel as the other systems in operation by the clients. (SO-04-1) The system supports Chrome, Firefox, and IE version 10 or above.

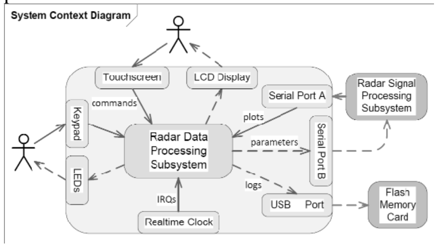
Requirements ID convention	User	System
Functional Requirements	UF-A	SF-A
Non-Functional Product Requirements	UP-01	SP-01
Non-Functional Organizational Requirements	UO-01	SO-01
Non-Functional External Requirements	UE-01	SE-01

Resolve Amalgamation:
The MentalCare system shall generate monthly management reports in 5 seconds. →
1)The MentalCare system shall generate monthly management reports
2)Each report generation should be done in 5 seconds
Reduce Ambiguity:
The MentalCare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.
1) A report shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
2) The MentalCare system shall generate the report for printing after 5:30pm on the last working day of the month.
3) Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.
Improve testability:
The MentalCare system should be easy to use by medical staff and should be organized in such a way that user errors are minimized.
→Medical staff shall be able to use all the system functions after four hours of training. After training, the average number of errors made by experienced users shall not exceed two per hour of system use.
System Requirements Specification (SRS)
Project manager: Use to plan the development process
Developer: Use to design/implement system components
Test engineer: Use to develop tests
Maintenance engineer: Use to understand the system

- System context: define the system's relationship to users or other related systems



- User interfaces: specify the required characteristics of each interface between the software product and its users. May required:
    - Screen formats, page or window layouts, content of report/menus/error messages, backup and recovery operation, user-initiated operations or function keys
- Software interfaces: specify the use of other required software systems and interfaces with other software systems:
    - Software name, specification number, version number, source. For each interface specify required message formats or if interface is documented provide reference to that document
    - If you are asked to implement a system in Java, would JRE be treated as other software system?
- Hardware interfaces & memory constraints: mainly applicable to embedded systems, specifying the logical characteristics of each interface between the software and the hardware elements of the system
    - Number of ports, instruction sets, required characteristic and constraints on primary & secondary memory. What devices are supported, how, and protocol



- Deployment requirements: Specify environment that is required for installation and operation of the software
    - Modifications to the customer's work area (screen size & resolution), any equipment customer need to buy, environment setup need to be done prior. Could be hardware/software specific
- Assumptions and dependencies: factors that affect the requirement (assumed user not color blind, network bandwidth 50Mbps minimum, etc.)
- Specific Requirements: <role><feature><benefit>

Example	Example
(SF-A-1) The system allows instructors to manage quiz questions & topics. (SF-B-1) The system supports up to 50 students to take quizzes concurrently. (SF-C-1) The system allows report preview and reformat. (SF-C-2) The system allows students to download report in PDF. (SF-D-1) Students can collect advisors' feedbacks on project reports. (SF-D-2) Students can collect advisors' feedbacks on presentations.	(SP-01-1) Figures that cannot fit into one page should be resized proportionally. (SP-01-2) Tables that cannot fit into one page are split and table header is repeated. (SP-02-1) Authentication is covered at both the application level and the transport level. (SP-03-1) The system interface has the same look and feel as the other systems in operation by the clients. (SO-04-1) The system supports Chrome, Firefox, and IE version 10 or above.

Use Case Diagram:

- <<include>>: an indispensable step of the base logic
- <<extend>>: extra logic, executed conditionally. When A extends B, A is done conditionally.

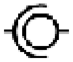
Use Case	#num: < the use case name as a short active verb phrase>
Goal in Context	< what the users want to accomplish by this use case>
Scope	< the system (or one of its subsystems) in which this use case belongs>
Level	< one of summary, primary task, subfunction>
Primary Actor	< the primary role who participates in this use case>
Preconditions	< the relevant situation description prior to the start of this use case>
Minimal Guarantee	< the relevant situation description upon an unsuccessful pass of the use case>
Success Guarantee	< the relevant situation description upon a successful pass of this use case>
Trigger	< the action upon the system that starts this use case>
Success Scenario	Action step
1	< the first step of the success scenario>
2	< the second step of the success scenario>
...	......
i	< a reference to another use case related by «include»>
...	......
n	< the last step of the success scenario>
Extension Step	Branching action (a step may have several branches)
2a	< the condition causing this branch>
	2a1: < a reference to another use case related by «extend»>
2b	< the condition causing this branch>
	2b1: < the first step of this branch>
	2b2: < the second step of this branch>
	......
	2bj: < the last step of this branch>

Aspect	Summary Level
Purpose	Used as "Table of contents." One for each (sub)system
Level test	Takes a long time to finish, maybe a few hours or days
Diagram indicators	Covers nearly all the use cases inside the system or subsystem boundary
Scenario steps	Generally has "user steps" only; often contains references to primary task level use cases

Aspect	Primary Task Level (User Goal Level)
Purpose	Used to describe base (backbone) logic
Level test	Takes one-sitting time to finish, maybe 2-20 min
Diagram indicators	May have direct links to actor(s); May «include», or be «extended» by, other use cases
Scenario steps	Generally lists "user-system" steps in an interleaving manner; often contains references to other use cases (via «include» or «extend»

Aspect	Subfunction Level
Purpose	Used to describe small reusable logic
Level test	Takes a very short time, say, less than 1 min
Diagram indicators	Has no direct link to the actor; its primary actor is the one linked to the base use case (by «extend» or «include»)
Scenario steps	Generally has only a few "user-system" steps in an interleaving manner

Interface: an abstraction that declares public operations only. Can be view as contractual agreement

- Service: Check-out by credit card
- Service pre-condition: valid card
- Service post-condition: transaction completed
- Service  client/user

```
//PeriodicTimer.java-----------------
public class PeriodicTimer implements Timer{
    private int priority;
    public void start(){
        //start the timer
    }
    public void stop(){
        //stop the timer
    }
    public void pause(){
        //pause the timer
    }
    public void resume(){
        //resume the timer
    }
}
```

```
//Timer.java-------------------
public interface Timer {
    public void start();
    public void stop();
    public void pause();
    public void resume();
}
```

Programming languages may have different ways to support interfaces

```
//TimeSensitiveTask.java-----------------
public class TimeSensitiveTask {
    private boolen taskInProgress = false;

    public void launchTimer(Timer t){
        t.start();
        taskInProgress = true;
    }
    public void withdrawTimer(Timer t){
        t.stop();
    }
    public void doWork(){
        while (taskInProgress){
            //make progress on task
        }
    }
}
```

Client-server: Service provider & service consumer at the architecture level	
Description	➢ The system functionality is organized into services. Services are available to all clients (regulated by accessibility control policy). ➢ Clients are users of these services.
When used	➢ when data in a shared database have to be accessed from a range of locations. ➢ When services have to be offered to an unknown amount of users (load balance by replicated servers).
Advantages	➢ Clients and servers can be deployed/executed on a single computer; or they can be distributed across a network. ➢ Support service-oriented system architecture.
Disadvantages	➢ Each service is a source of single point of failure (DoS attacks or server failure). ➢ System performance may be unpredictable because it also depends on the network. ➢ Management issues may exist if servers are owned by different organizations.

Name	Layered architecture
Description	➢ A system is organized into layers, each of which provides a set of services. ➢ A layer provides services to the layer above it. The lowest-level layer represents core services that are likely to be used throughout the system.
When used	➢ when building new facilities on top of existing systems; ➢ when the development is spread across several teams with each team being responsibility for a layer of functionality; ➢ when there is a requirement for multi levels of filtering or processing (say, security).
Advantages	➢ Flexibility: allows replacement of a layer so long as the inter-layer interface (API) is maintained. ➢ Separation of concerns.
Disadvantages	➢ In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. ➢ Performance can be a problem because of multiple levels of processing of a service request.

Name	Repository
Description	➢ All data in a system is managed in a central repository that is accessible to all system components. ➢ Components do not interact directly, only through the repository.
When used	➢ when you have a system in which large volumes of information are generated that has to be stored for a long time. ➢ In data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	➢ Components can be independent—they do not need to know of the existence of other components. ➢ Changes made by one component can be propagated to all components. ➢ All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	➢ The repository is a single point of failure and/or communication bottleneck so problems in the repository affect the whole system. ➢ Management issues when the repository is distributed/replicated over networks.