SE-2002
SOFTWARE DESIGN AND ARCHITECTURE

RUBAB JAFFAR

RUBAB.JAFFAR@NU.EDU.PK

Introduction

Software Processes, UP and UML

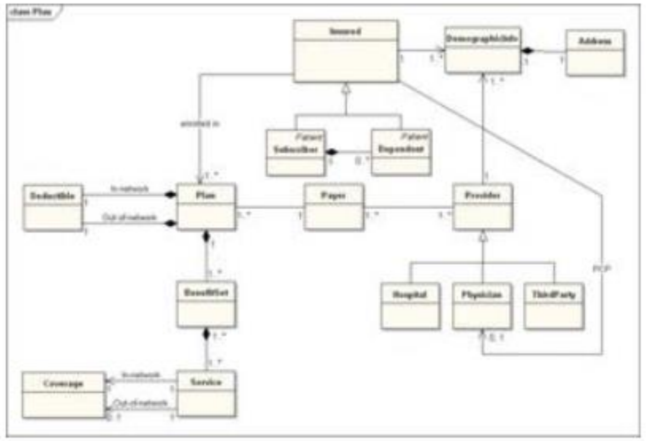
Domain Model

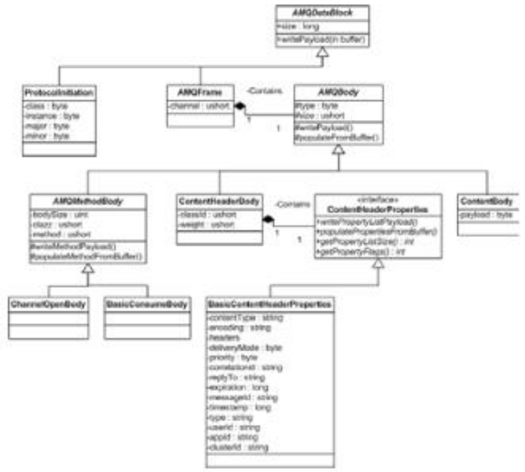
Lecture # 4, 5, 6

TODAY'S OUTLINE

- Software Process
- Software Development approaches
- Process models
- Unified Process
- Use case model
- UML

OOA/OOD





Analyze the system

Model the system

Design the software

ANALYSIS AND DESIGN

- Analysis is the investigation of the problem - what are we trying to do?
  - Here is where use cases are created and requirements analysis are done
- Design is a conceptual solution that meets the requirements – how can we solve the problem
  - Note: Design is not implementation
  - UML diagrams are not code (although some modeling software does allow code generation)
- Object-oriented analysis: Investigate the problem, identify and describe the objects (or concepts) in the problem domain
  - Also, define the domain!
- Object-oriented design: Considering the results of the analysis, define the software classes and how they relate to each other
  - Not every object in the problem domain corresponds to a class in the design model, and viceversa
- Where do we assign responsibilities to the objects? Probably a little in both parts

THE UML

WHAT IS UML?

- The Unified Modeling Language (UML) is a language for **specifying, visualizing, constructing, and documenting** the artifacts of software systems, as well as for business modeling and other non-software systems.
- Visual
- We will use UML to sketch out our systems
- We will explore the details of UML diagramming
- For now, understand that UML is a language – it is used to communicate information
- We will use UML to describe the problem domain, describe the activities that occur, and eventually describe the software classes
- Since it is a language, UML has specific rules, and we will see these later in the course
- You need to be able to read UML diagrams, as well as create them

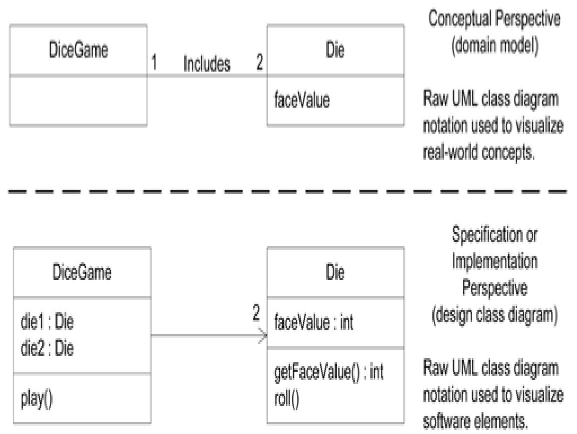
THREE WAYS TO APPLY UML

- UML as a Sketch
- UML as a Blueprint
- UML as a Programming Language

What does Agile Modeling emphasize?

THE PERSPECTIVES TO APPLY UML

- Conceptual Perspective
- Specification (Software) Perspective
- Implementation (Software) Perspective


Conceptual Perspective (domain model)
DiceGame
1
Includes
2
Die
faceValue
Raw UML class diagram notation used to visualize real-world concepts.
Specification or Implementation Perspective (design class diagram)
DiceGame
die1 : Die
die2 : Die
play()
2
Die
faceValue : int
getFaceValue() : int
roll()
Raw UML class diagram notation used to visualize software elements.

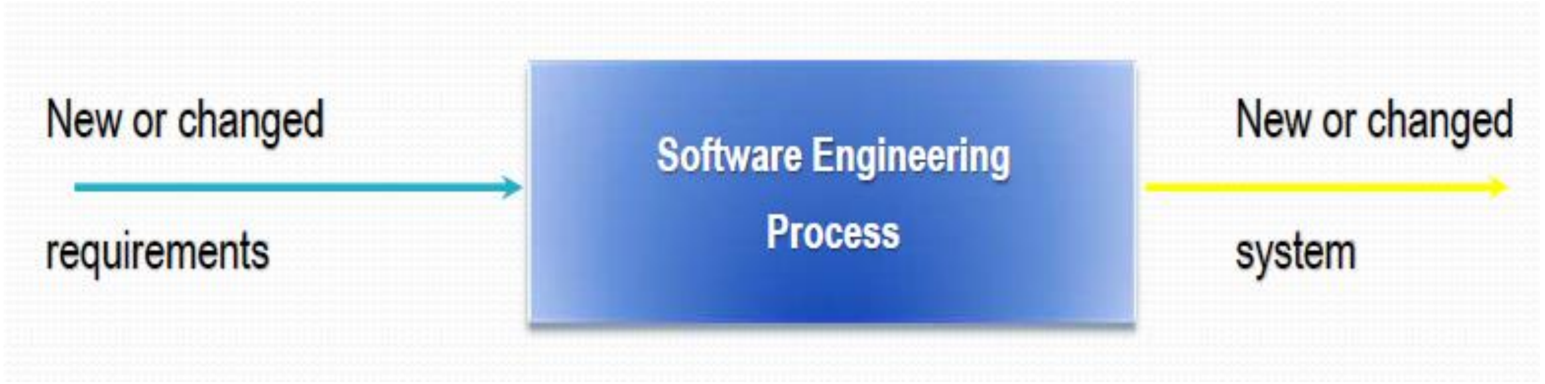
- Conceptual Class
- Specification Class
- Implementation Class

UML AND “SILVER BULLET” THINKING

DO UML DIAGRAMS REALLY MAKE THINGS BETTER?

PROCESS

Defines Who is doing What, When to do it, and How to reach a certain goal.



- Workers, the ‘who’
- Activities, the ‘how’
- Artifacts, the ‘what’
- Workflows, the ‘when’

WHAT IS A PROCESS MODEL ?

It is a description of

i) <u>what tasks need to be performed</u> in

ii) <u>what sequence</u> under

iii) <u>what conditions</u> by

iv) <u>whom</u> to achieve the "desired results."

Why Process Model?

Provide "guidance" for a systematic <u>coordination</u> and <u>controlling</u> of

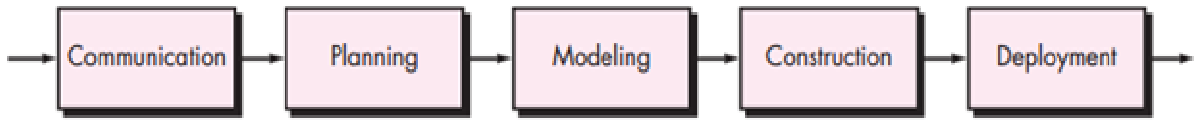
a) the tasks and of

b) the personnel who perform the tasks

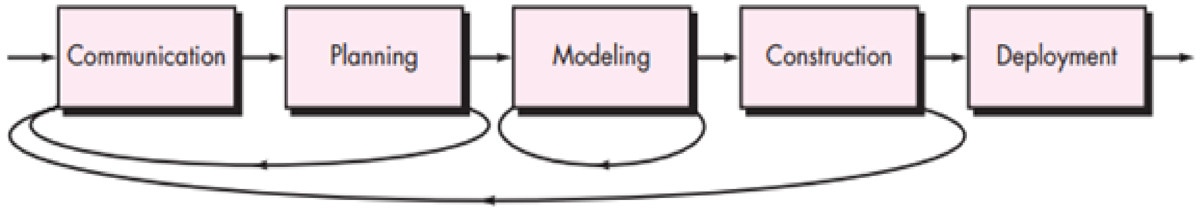
Note the key words: <u>coordination/control</u> , <u>tasks</u>, <u>people</u>

GENERIC PROCESS FLOWS

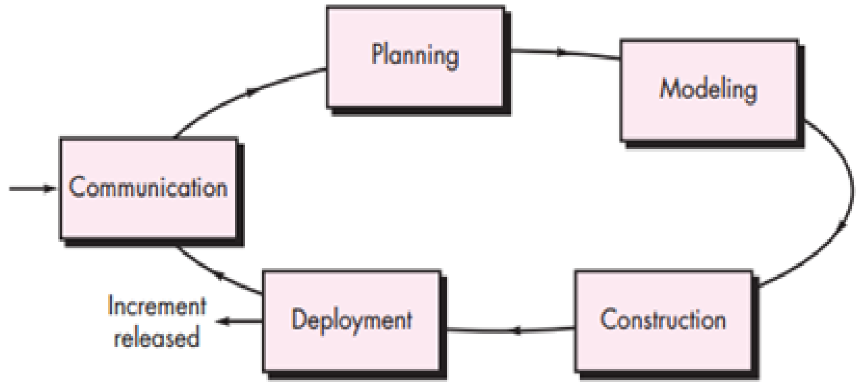
- Linear process flow
- Iterative process flow
- Incremental process flow



(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow

?????

- Can You recall some process models that have been studied in ISE and what about their process flow?

ITERATIVE, EVOLUTIONARY AND AGILE

- **Iterative**: done over many iterations (X weeks)
- **Evolutionary**: evolves over time (iteration time)
- **Agile**: Delivering the product in incremental chunks

WHAT IS AGILE METHODOLOGY?

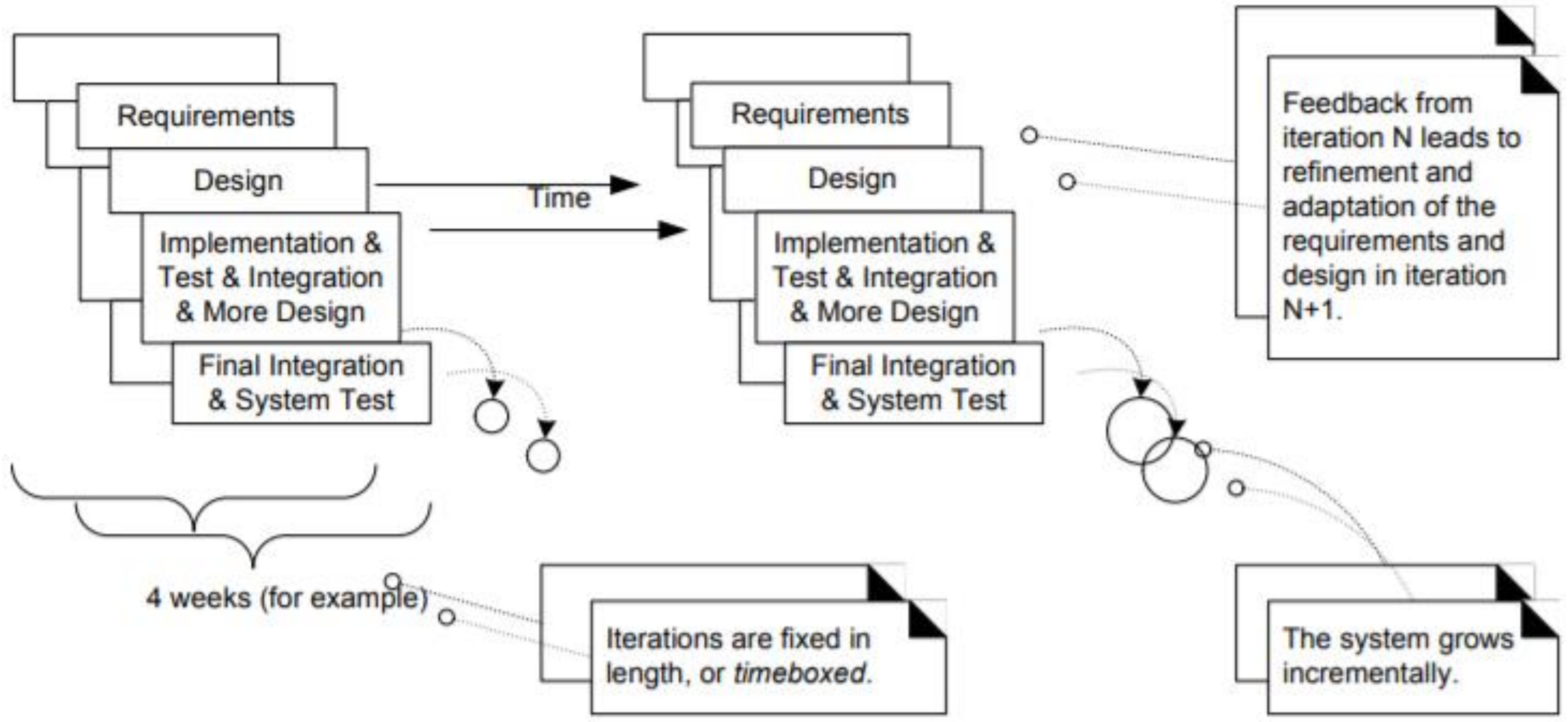
- Agile methodologies are approaches to product development that are aligned with the values and principles described in the Agile Manifesto for software development.

- Agile methodologies aim to deliver the right product, with incremental and frequent delivery of small chunks of functionality, through small cross-functional self-organizing teams, enabling frequent customer feedback and course correction as needed.
- The Agile Manifesto:
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan

ITERATIVE DEVELOPMENT

- Suppose you were assigned to write a movie script for a company. They know what they want, in terms of what kind of movie, how long, setting, etc., but they need you to fill in the details. How would you do it?
- You could spend a lot of time talking to them, getting as much information as possible, and then write the script and send it to them and hope you nailed it …
- Risky: Chances of getting it all right are slim, and if you missed you need to go back and start making changes and edits, which can be complicated if you want the story line to work
- You could also start with a draft, and send the incomplete version to them for feedback They would understand that this is not finished, but just a draft.
- They provide feedback, you add more details, and the cycle continues Eventually, you end up with the finished product that is close to what they wanted
- This is how iterative development works

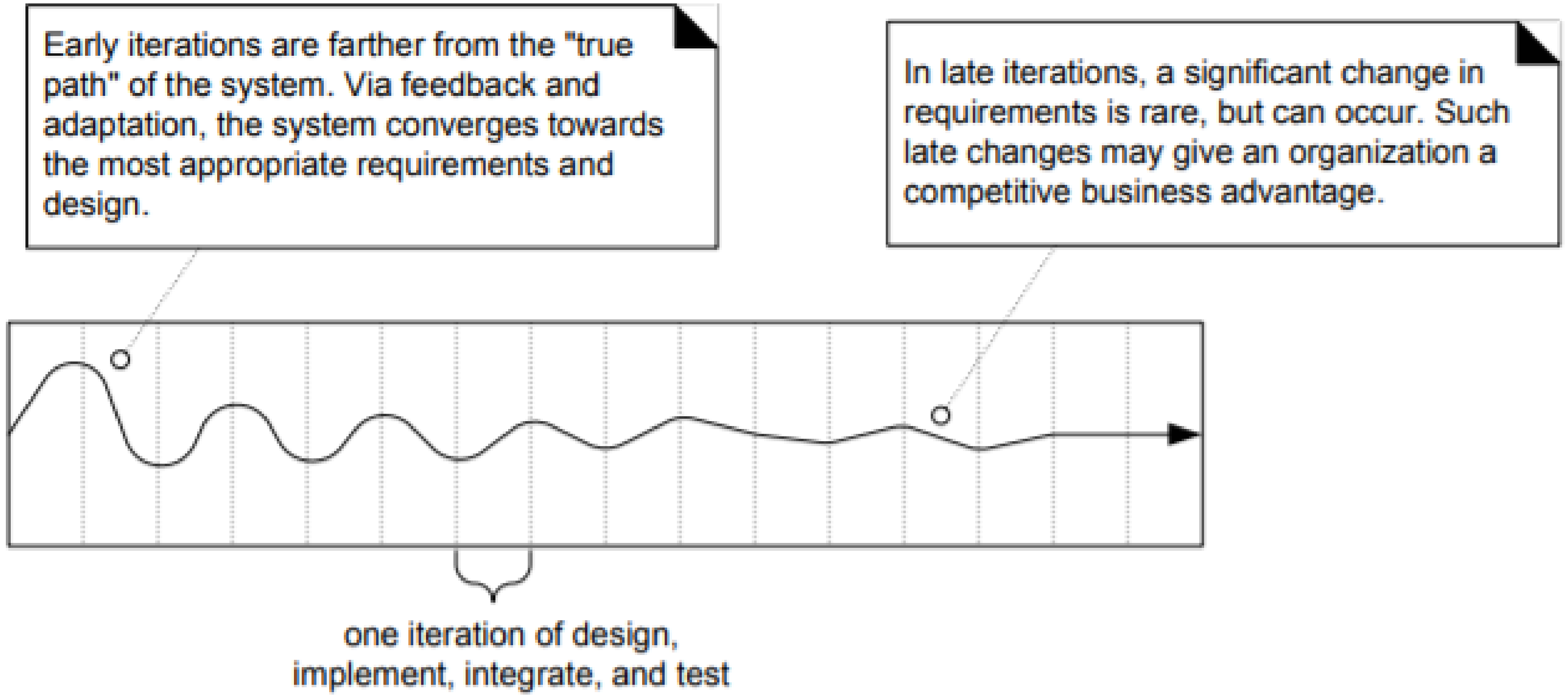
ITERATIVE AND EVOLUTIONARY DEVELOPMENT


Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Time
Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.
4 weeks (for example)
Iterations are fixed in length, or *timeboxed*.
The system grows incrementally.

HOW TO HANDLE CHANGE IN AN ITERATIVE PROJECT?

- Do you specify, freeze, and "sign off" on a set of requirements?
- Iterative development is based on an attitude of embracing change and adaptation as unavoidable
- Rather than try to “freeze” the requirements so code can be developed to rigid design, accept the fact that change will occur and use it in the development process
- Maybe the customer changes mind after seeing early partial system builds
- This has become very popular because it reflects the way the real world works
- Also, coding technology has enhanced code generation methods
- OOA/OOD is critical to this approach, since having well defined objects makes it easier to add to or modify the system
- Note that this does not mean there are no requirements, or that changes should be encouraged
- Beware of “feature creep” – we will see more later
- This is structured change, not chaos

ITERATIVE COVERAGE (FEEDBACK AND ADAPTATION)


Early iterations are farther from the "true path" of the system. Via feedback and adaptation, the system converges towards the most appropriate requirements and design.
In late iterations, a significant change in requirements is rare, but can occur. Such late changes may give an organization a competitive business advantage.
one iteration of design, implement, integrate, and test

BENEFITS OF ITERATIVE DEVELOPMENT

- Less project failure, better productivity, lower defect rates
- Early mitigation of high risks (technical, requirements, objectives, usability, and so forth)
- Early visible progress
- Early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders
- Managed complexity; the team is not overwhelmed by "analysis paralysis" or very long and complex steps
- The learning within an iteration can be methodically used to improve the development process itself, iteration by iteration

ITERATION LENGTH & TIME-BOXING

- How long should your iteration be?
- **Timeboxed**: Fixed in Length
- So what if you can’t complete the task by the scheduled date?
- Normal Length: 3-6 weeks

WATERFALL THINKING

HOW DO YOU KNOW IF YOU'RE FOLLOWING THE WATERFALL THINKING IN AN ITERATIVE PROJECT?

WATERFALL LIFECYCLE

- Probably an attempt to apply standard product development methods to software
- Fully define requirements (and usually design) before coding
- Why is Waterfall so prone to fail?
- Software development is not mass production – highly unlikely that all requirements are fully understood up front
- Requirement are also not as stable as we would like, especially for large complex projects
- Change Request process works, but can be cumbersome and slow
- It is possible to change requirements later in the project for Waterfall, but it is hard and slow
- Try to avoid combining this approach with an iterative approach
- Don't try to identify all use cases or do complete OOA before coding can start
- Software design and implementation becomes more complex as understanding of the system increases
-

TYPES OF FEEDBACK

- Feedback from early development/client demos
  - To refine requirements
- Feedback from tests and developers
  - To refine design and models
- Feedback from the progress of the team
  - To refine schedules and estimates
- Feedback from the client and the marketplace
  - To re-prioritize features for next iteration

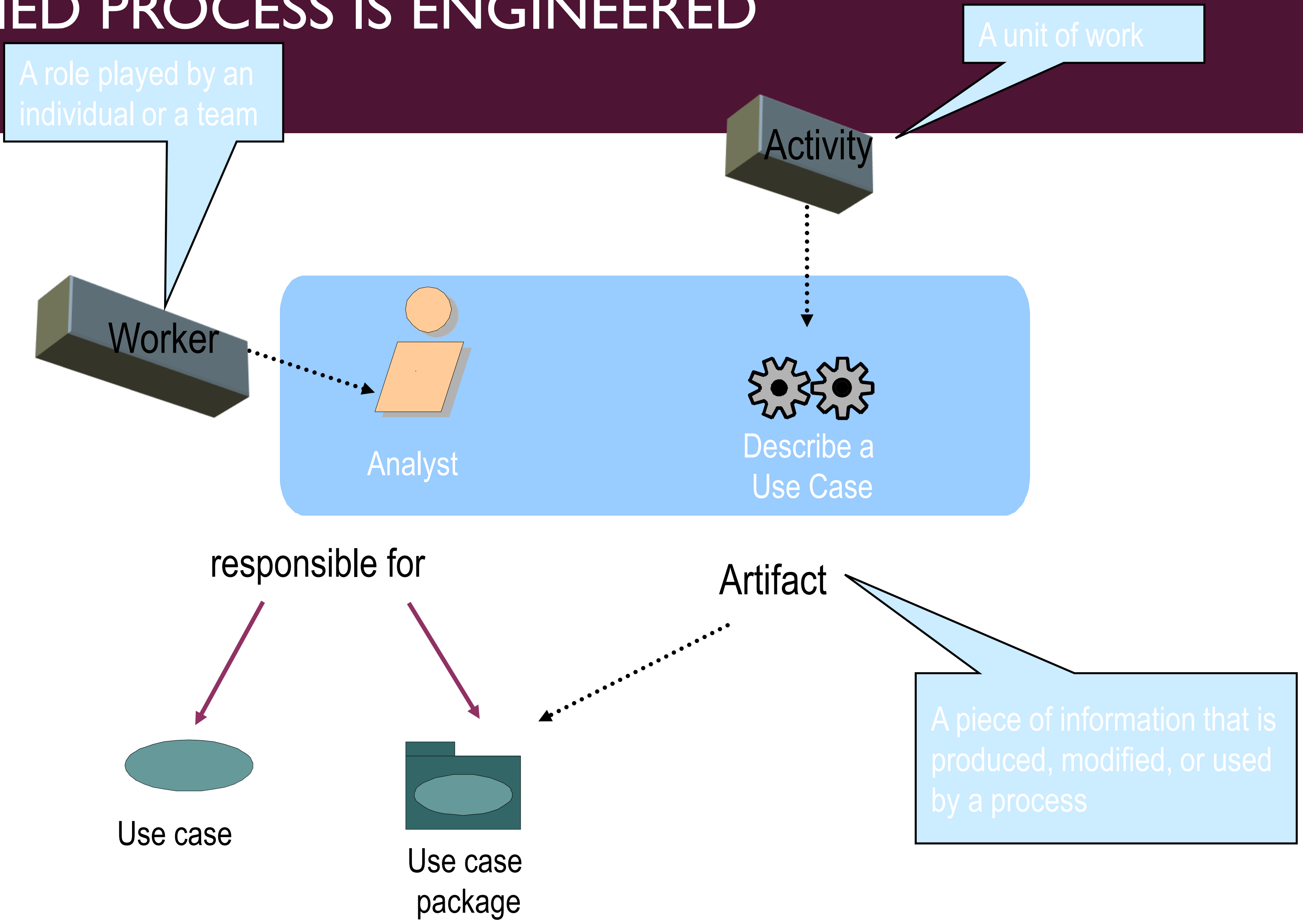
UNIFIED PROCESS MODEL

- A process model that was created 1997 to give a framework for Object-oriented Software Engineering
- Iterative, incremental model to adapt to specific project needs
- Risk driven development
- Combining spiral and evolutionary models
- **2D process : phases and workflows**
- **Utilizes Miller's Law**

UNIFIED PROCESS

- The Unified Process is not simply a process, but rather an extensible framework which should be customized for specific organizations or projects.
- The Rational Unified Process is, similarly, a customizable framework. As a result, it is often impossible to say whether a refinement of the process was derived from UP or from RUP, and so the names tend to be used interchangeably.
- A standardized approach to analysis and design helps to ensure that all necessary tasks are understood and completed in software development.
- Requirements analysis and OOAD needs to be presented and practiced in the context of some development process.
- An agile approach to well-known UP is used as an iterative development

THE UNIFIED PROCESS IS ENGINEERED


A role played by an individual or a team
A unit of work
Activity
Worker
Analyst
Describe a Use Case
responsible for
Artifact
A piece of information that is produced, modified, or used by a process
Use case
Use case package

BUILDING BLOCKS OF UP

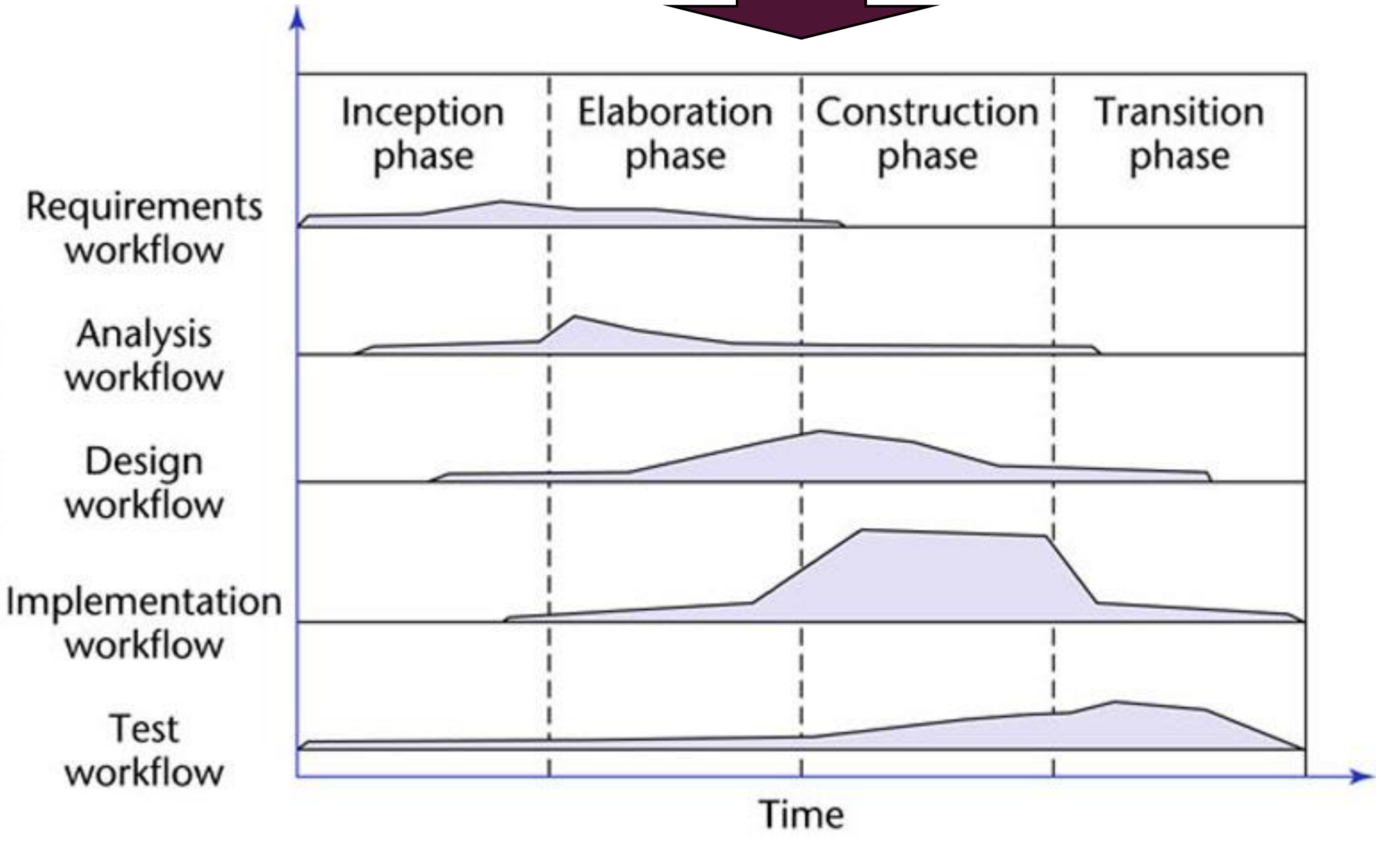
- All aspects of the Rational Unified Process are based on a set of building blocks, which are used to describe what should be produced, who is in charge of producing it, how production will take place, and when production is complete.
- These four building blocks are:

- Workers, the 'Who': The behavior and responsibilities of an individual, or a group of individuals together as a team, working on any activity in order to produce artifacts.
- Activities, the 'How': A unit of work that a worker has to perform. Activities should have a clear purpose, typically by creating or updating artifacts.
- Artifacts, the 'What': An artifact represents any tangible output that emerges from the process; anything from new code functions to documents to additional life cycle models.
- Workflows, the 'When': Represents a diagrammed sequence of activities, in order to produce observable value and artifacts.

THE PHASES/WORKFLOWS OF THE UNIFIED PROCESS

- Phase is Business context of a step

Workflow



THE UNIFIED PROCESS (UP)

- A software development process describes an approach to building, deploying, and maintaining software.

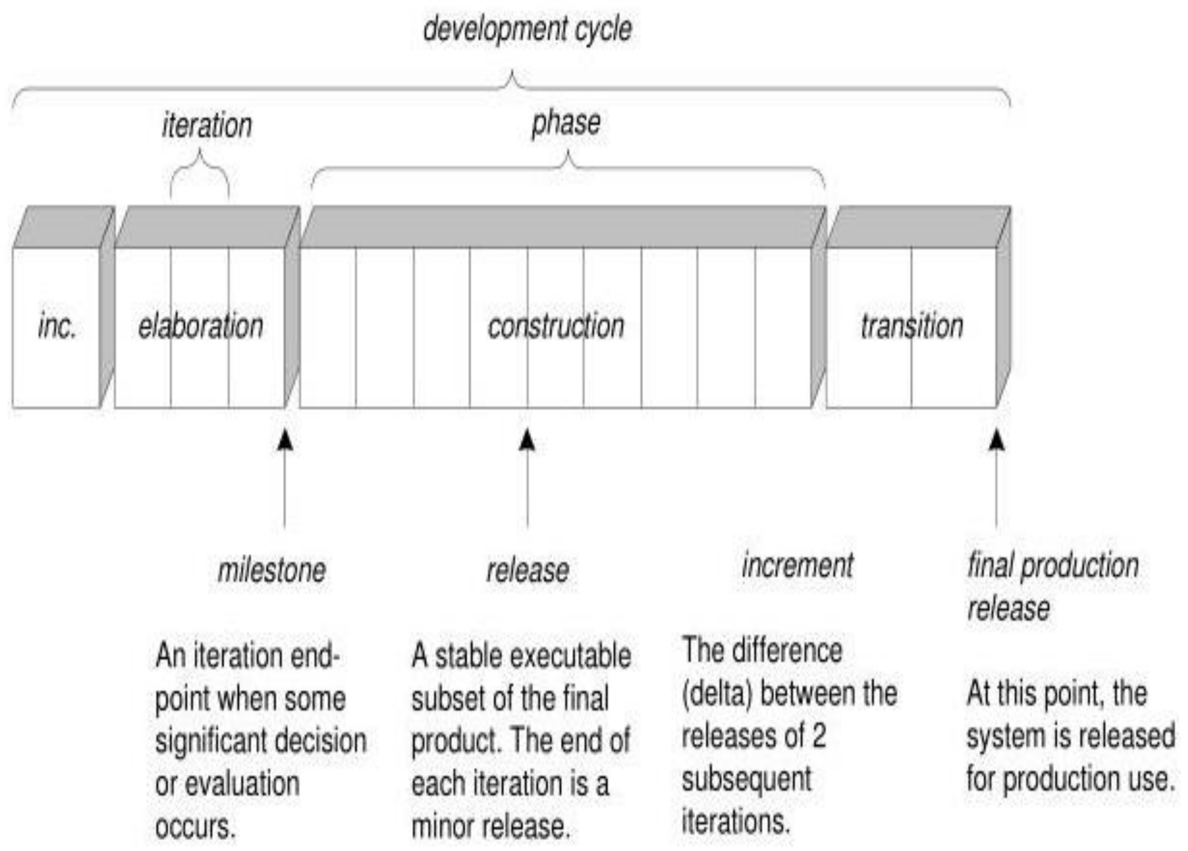
- UP has emerged as a popular and effective iterative software development process for building OO systems.
  - Rational Unified Process (RUP) is a modified version of the Unified Process, which was modified by Rational Software, is widely practiced and adopted by our industry.

UP – THE KEY PRACTICES

- Tackle high-risk and high-value issues in early iterations
- Continuously engage users for evaluation and feedback, requirements evolution
- Build a cohesive, core architecture early on
- Continuously verify quality: test early and often, and realistically!
- Apply use cases where appropriate
- Utilize visual modeling (UML)
- Carefully manage requirements
- Practice change request and configuration management

THE UP PHASES

Inception: approximate the vision, business case, scope, vague estimates
Elaboration: Refined vision, iterative implementation of the core architecture,
resolution of high risks, identification of most requirements and scope, realistic
estimates
Construction: Iterative implementation of the remaining low risk and easier
elements, preparation for deployment
Transition: Beta testing, deployment


development cycle
iteration
phase
inc.
elaboration
construction
transition
milestone
release
increment
final production release
An iteration end-point when some significant decision or evaluation occurs.
A stable executable subset of the final product. The end of each iteration is a minor release.
The difference (delta) between the releases of 2 subsequent iterations.
At this point, the system is released for production use.

UP PHASES

- **Inception**
- Approximate vision, business case, scope, vague estimates.
- Note: Inception is not a requirements phase; rather, it is a kind of feasibility phase, where just enough investigation is done to support a decision to continue or stop.

UP PHASES

- **Elaboration**
- Refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates

- Note: Elaboration is not the requirements or design phase; rather, it is a phase where the core architecture is iteratively implemented, and high risk issues are mitigated.

UP PHASES

- **Construction**
- Iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.
- Note: Construction is not the implementation phase

UP PHASES

- **Transition**

Beta tests, deployment

GOALS AND FEATURES OF EACH ITERATION

- Slowly chip away at the project risks:
  - performance risks
  - integration risks (different vendors, tools, etc.)
  - conceptual risks (hunt out analysis and design flaws)
- Perform a "miniwaterfall" project that ends with a delivery of something tangible in code.
- Each iteration is risk-driven.
- The result of a single iteration is an incremental improvement.

INCEPTION

FIRST PHASE OF THE UNIFIED PROCESS

Inception, Requirements, Use Cases

- Inception – what is it?
- How to analyze requirements in iterative development
- How to identify and write Use Cases
- How to apply tests to identify suitable Use Cases
- How to develop Use Cases in iterative development

INCEPTION

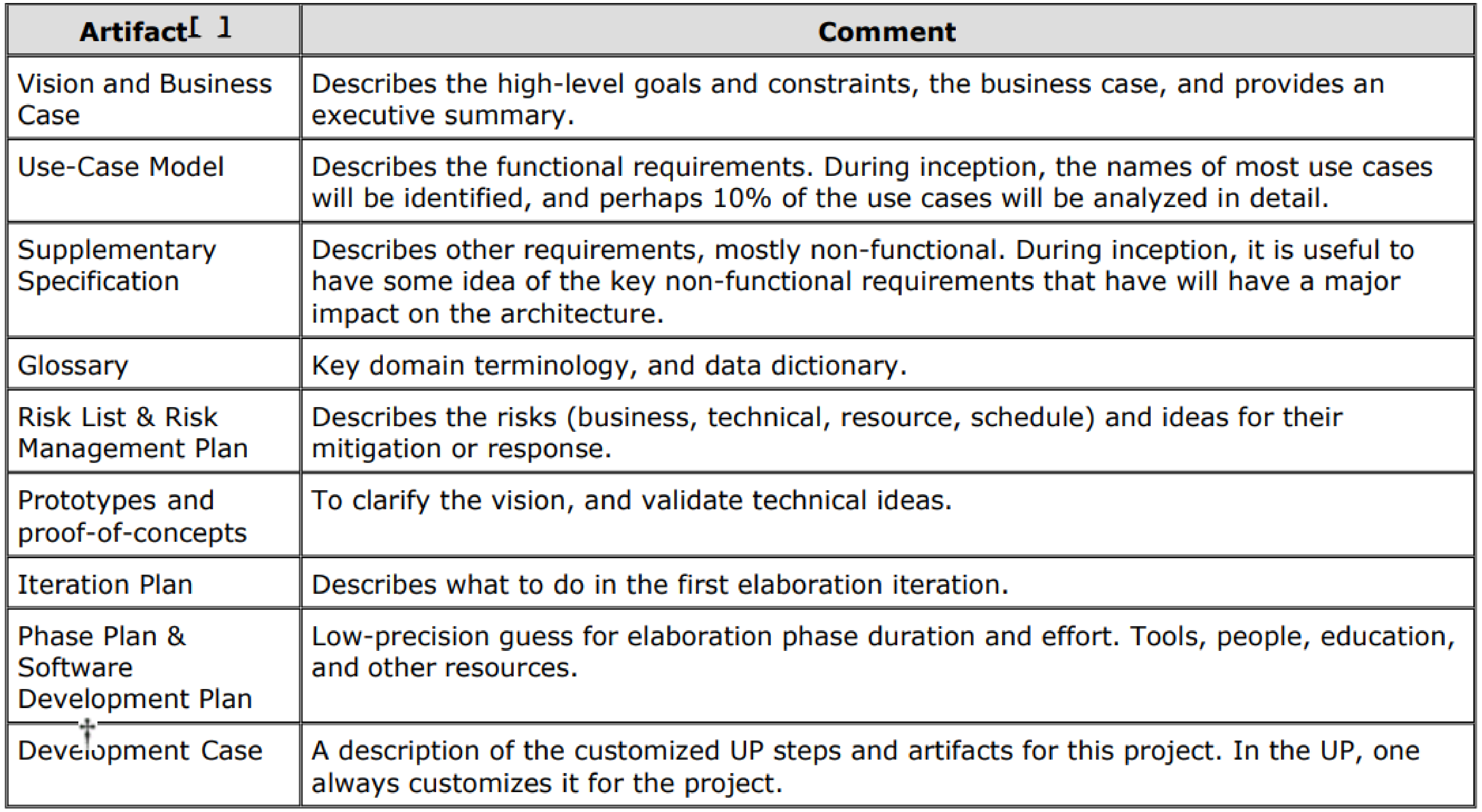
- **Inception is the initial short step to establish a common vision and basic scope for the project.**
- It will include analysis of perhaps 10% of the use cases, analysis of the critical non-functional requirement, creation of a business case, and preparation of the development environment so that programming can start in the following elaboration phase.
- Main questions that are often asked:
- What is the overall vision and business case for the project?
- Is it feasible?
- Buy or build?
- Rough cost estimate (order of magnitude)
- Go, no go
- We do not define all of the requirements in Inception!
- Perhaps a couple of example requirements, use cases
- We are not creating a project plan at this point

INCEPTION

- **Inception in one sentence:**
- Envision the product scope, vision, and business case.

- **The main problem solved in one sentence:**
- Do the stakeholders have basic agreement on the vision of the project, and is it worth investing in serious investigation?

INCEPTION ARTIFACTS



Artifact[ ]	Comment
Vision and Business Case	Describes the high-level goals and constraints, the business case, and provides an executive summary.
Use-Case Model	Describes the functional requirements. During inception, the names of most use cases will be identified, and perhaps 10% of the use cases will be analyzed in detail.
Supplementary Specification	Describes other requirements, mostly non-functional. During inception, it is useful to have some idea of the key non-functional requirements that have will have a major impact on the architecture.
Glossary	Key domain terminology, and data dictionary.
Risk List & Risk Management Plan	Describes the risks (business, technical, resource, schedule) and ideas for their mitigation or response.
Prototypes and proof-of-concepts	To clarify the vision, and validate technical ideas.
Iteration Plan	Describes what to do in the first elaboration iteration.
Phase Plan & Software Development Plan	Low-precision guess for elaboration phase duration and effort. Tools, people, education, and other resources.
Development Case	A description of the customized UP steps and artifacts for this project. In the UP, one always customizes it for the project.

IDENTIFYING REQUIREMENTS

- A **requirement** may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification
- There are two types of requirements:
  - **User Requirements:**
    - Written for customers.
    - Often written in natural language (no technical details)
  - **System Requirements:**
    - Written for developers
    - Detailed functional and non-functional requirements discussed.
    - Include clear implementation details
    -

SYSTEM STAKEHOLDERS

- End users
- System managers
- System owners
- External stakeholders

FUNCTIONAL REQUIREMENTS:

- Statements of services the system should provide
- How the system should react to particular inputs and in particular situations.
- May state what the system should not do

NON-FUNCTIONAL REQUIREMENTS

- Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Similar to Quality attributes
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless

NON-FUNCTIONAL REQUIREMENTS

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

USE CASES

ACTORS, SCENARIOS & USE CASES

- An **actor** is something with behavior, such as a person (identified by role), computer system, or organization; for example, a cashier.
- A **scenario** is a specific sequence of actions and interactions between actors and the system; it is also called a **use case instance**.
- A **use case** is a collection of related success and failure scenarios that describe an actor using a system to support a goal.

USE CASE MODEL

- Use cases are text documents, not diagrams, and **use-case modeling is primarily an act of writing text, not drawing diagrams.**

TYPES OF ACTORS

- **Primary actor** has user goals fulfilled through using services of the system. For example, the cashier.

- Why identify?
- To find user goals, which drive the use cases.

TYPES OF ACTORS

- **Supporting actor** provides a service (for example, information) to the system. The automated payment authorization service is an example. Often a computer system, but could be an organization or person.

- Why identify?
- To clarify external interfaces and protocols.

TYPES OF ACTORS

- **Offstage actor** has an interest in the behavior of the use case, but is not primary or supporting; for example, a government tax agency.

- Why identify?
- To ensure that all necessary interests are identified and satisfied. Offstage actor interests are sometimes subtle or easy to miss unless these actors are explicitly named.

COMPONENTS OF USE CASE DIAGRAM

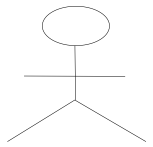
- Actors
- Use Case
- Relationship
- Boundary

ACTORS

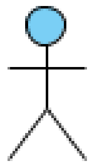
Actors

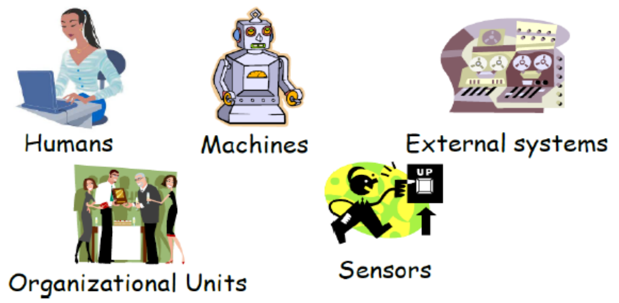
The people or systems that provide or receive information from the system;
Could be human beings, other systems, timers and clocks or hardware devices.

Actors that stimulate the system and are the initiators of events are called primary actors (active)
Actors that only receive stimuli from the system are called secondary actors (passive)



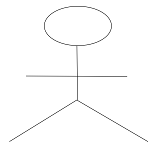
Notation





ACTORS

Actors

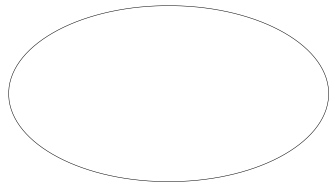


- Who/what will be interested in the system?
- Who/what will want to change the data in the system?
- Who/what will want to interface with the system?
- Who/what will want information from the system?

COMPONENTS OF USE CASE MODEL

- Use Case
  - Define the functionality that is handled by the system.
  - Each use case specifies a complete functionality (from its initiation by an actor until it has performed the requested functionality).
  - Describes the interactions between various actors and the system.

- Notation

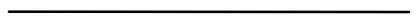


USE CASE - RELATIONSHIPS AND ITS TYPES

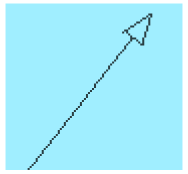
- Relationships
  - Represent communication between actor and use case
- 4 types of relationships
  - Association relationship
  - Generalization relationship
    - Generalization relationship between actors
    - Generalization relationship between use cases
  - Include relationship between use cases
  - Extend relationship between use cases

USE CASE - RELATIONSHIPS AND ITS TYPES

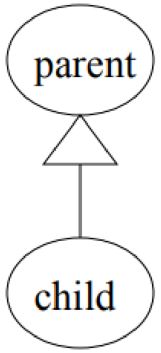
- Association relationship: Represent communication between actor and use case
  - Often referred to as a communicate association
  - use just a line to represent
  - Notation

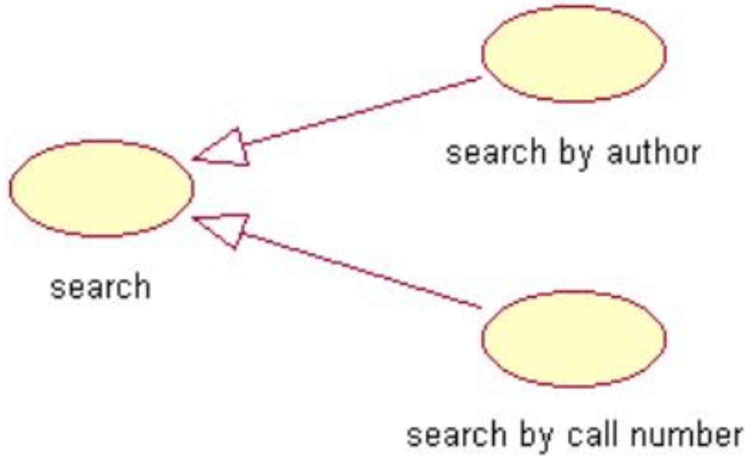


USE CASE - RELATIONSHIPS AND ITS TYPES



- Generalization:
- The child use case inherits the behaviour and meaning of the parent use case.
- The child may add to or override the behaviour (
- Notation:





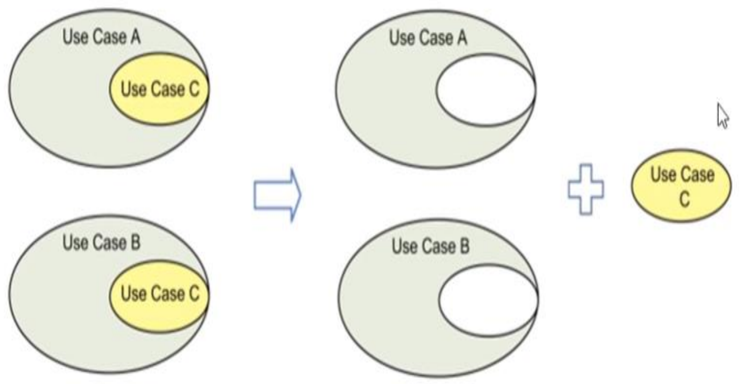
USE CASE - RELATIONSHIPS AND ITS TYPES

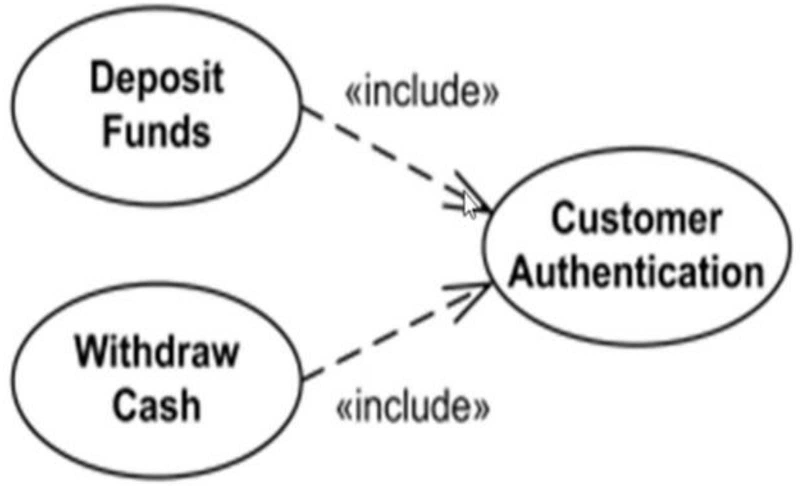
- Generalization relationship between actors
  - actor generalization refers to the relationship which can exist between two actors and which shows that one actor (descendant) inherits the role and properties of another actor (ancestor).
- Generalization relationship between use cases
  - use case generalization refers to the relationship which can exist between two use cases and which shows that one use case (child) inherits the structure, behavior, and relationships of another use case (parent).

USE CASE - RELATIONSHIPS AND ITS TYPES

- **Include**
  - Specifies that the source use case explicitly incorporates the behavior of another use case at a location specified by the source
  - The include relationship adds additional functionality not specified in the base use case.
  - <<include>> is used to include common behaviour from an included use case into a base use case
  - Notation *<<include>>* --------------->

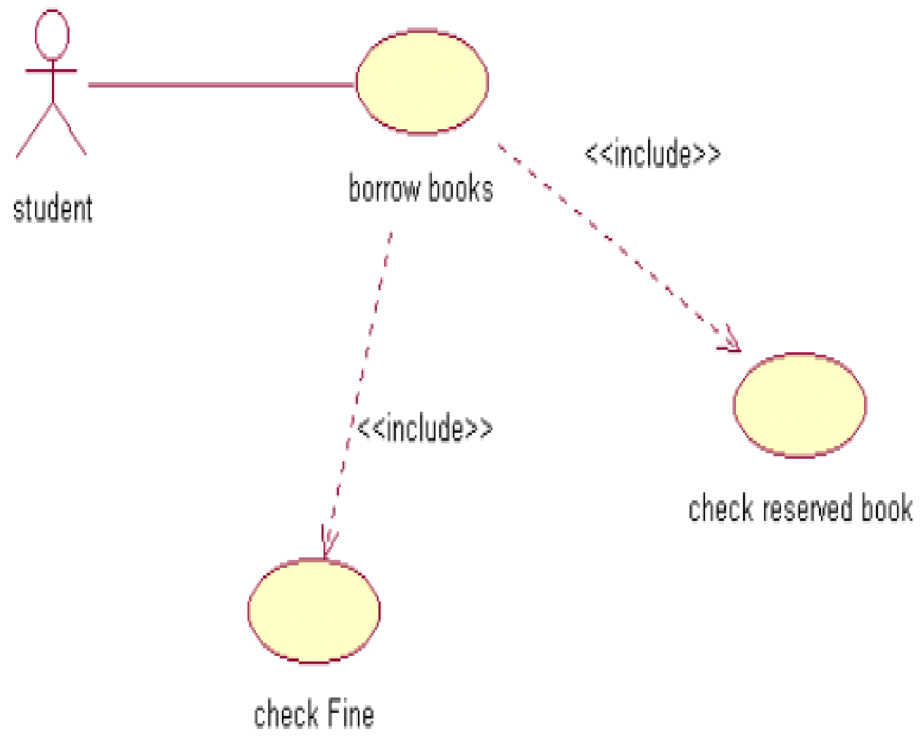
CONCEPT OF INCLUDE
PRACTICAL EXAMPLE OF INCLUDE





<<INCLUDE>>

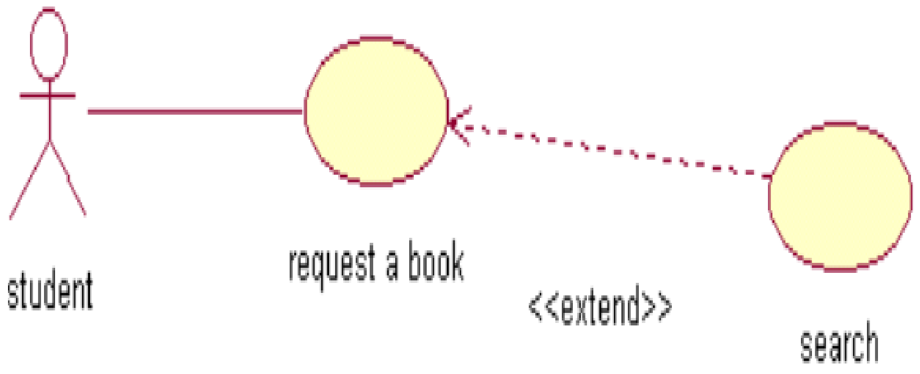
- An include relationship connects a base use case (i.e. borrow books) to an inclusion use case (i.e. check Fine).
- An include relationship specifies how behaviour in the inclusion use case is used by the base use case.



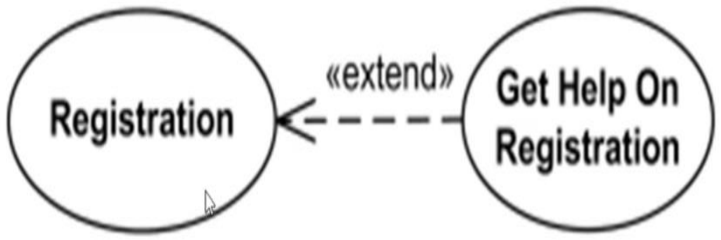
USE CASE - RELATIONSHIPS AND ITS TYPES

- **Extend**
  - Specifies that the target use case extends the behavior of the source.
  - The extend relationships shows optional functionality or system behaviour.
  - <<extend>> is used to include optional behaviour from an extending use case in an extended use case.
  - Notation *<<extend>>* <-----------------

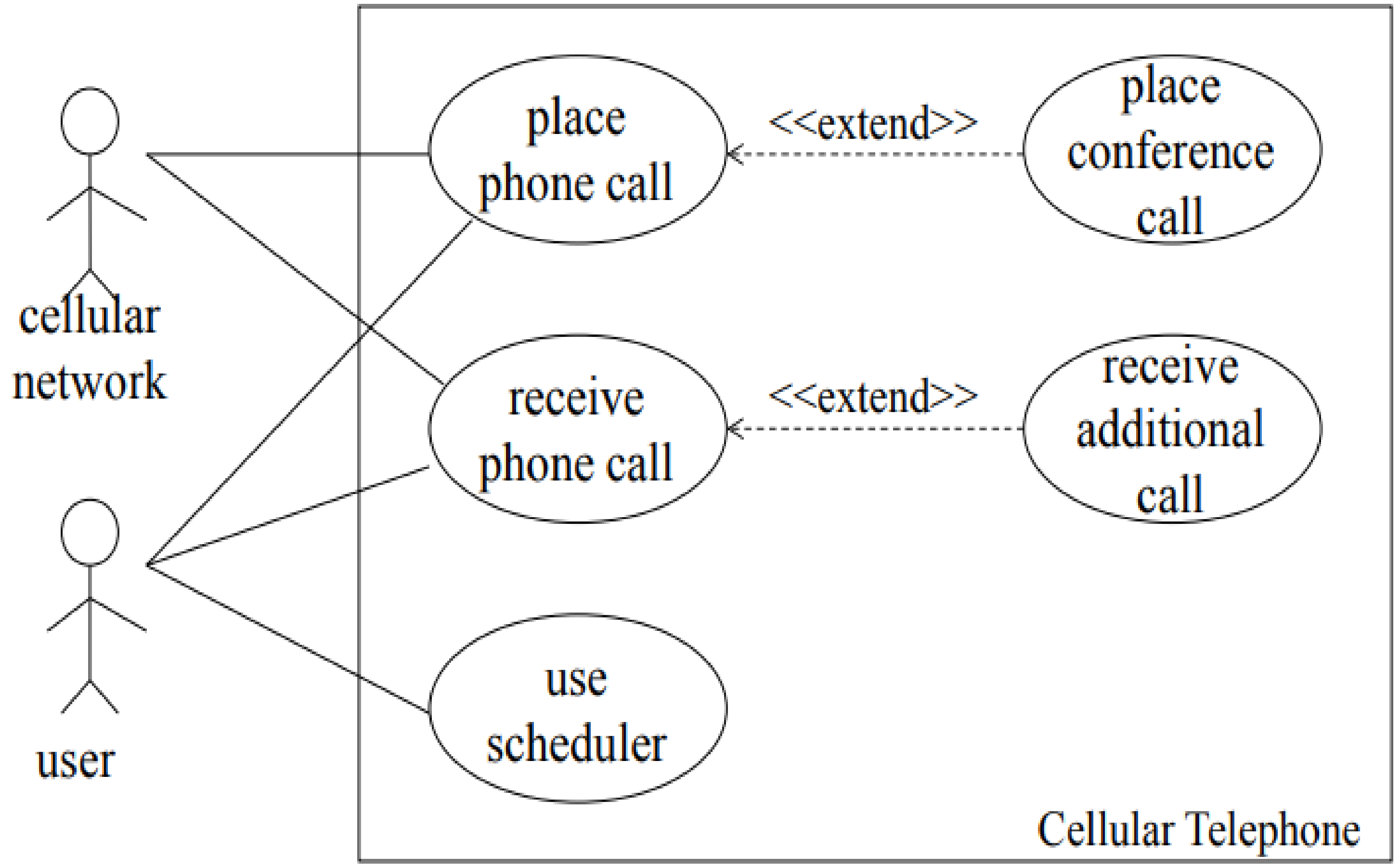
<<EXTEND>>



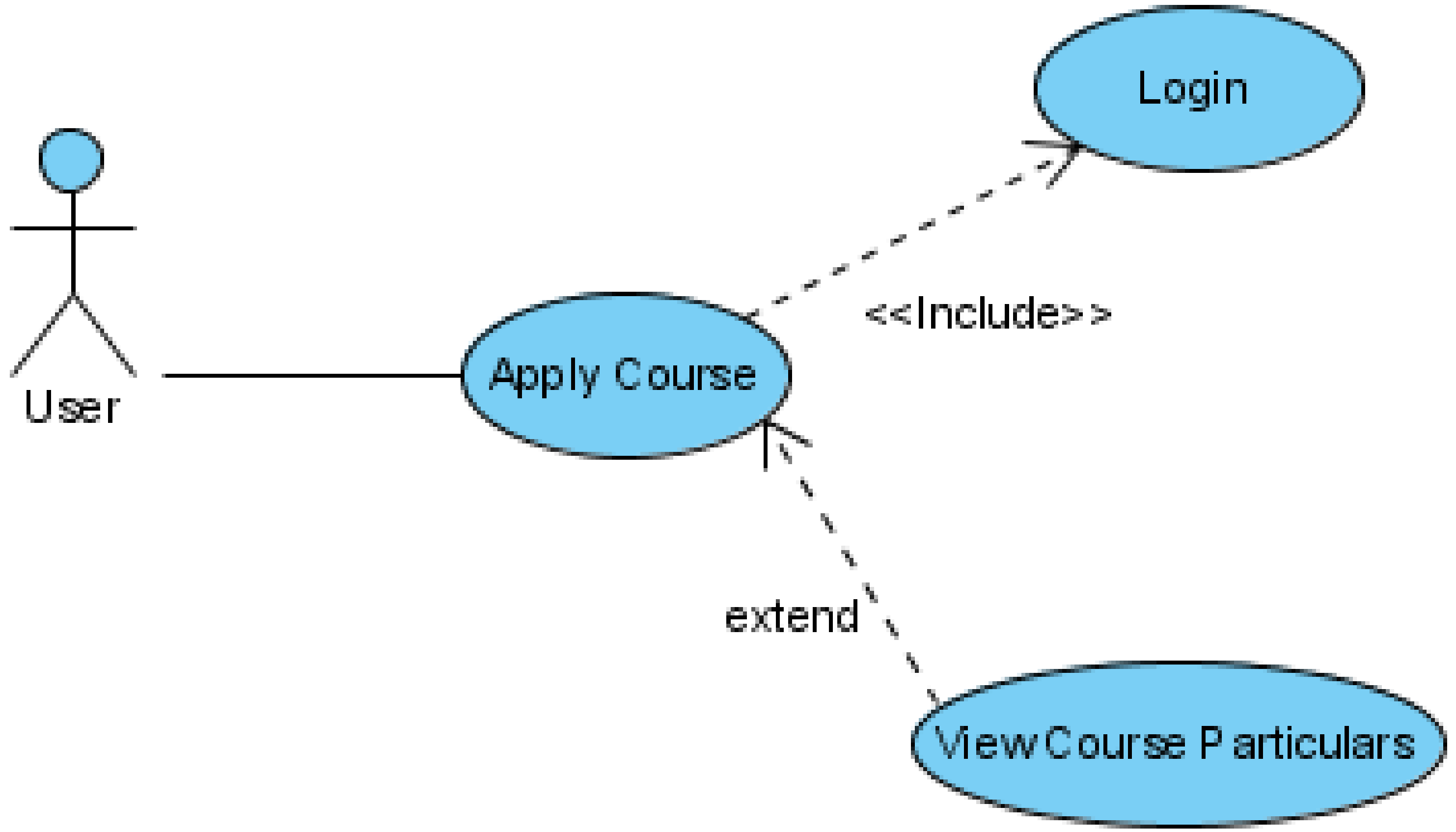
- The extend relationship is in between Request a book and Search.
- If the student desires, he/she can search the book through the system.
- However, the student may only Request a book through the system without searching the book if the student knows the call number.



CELLULAR TELEPHONE

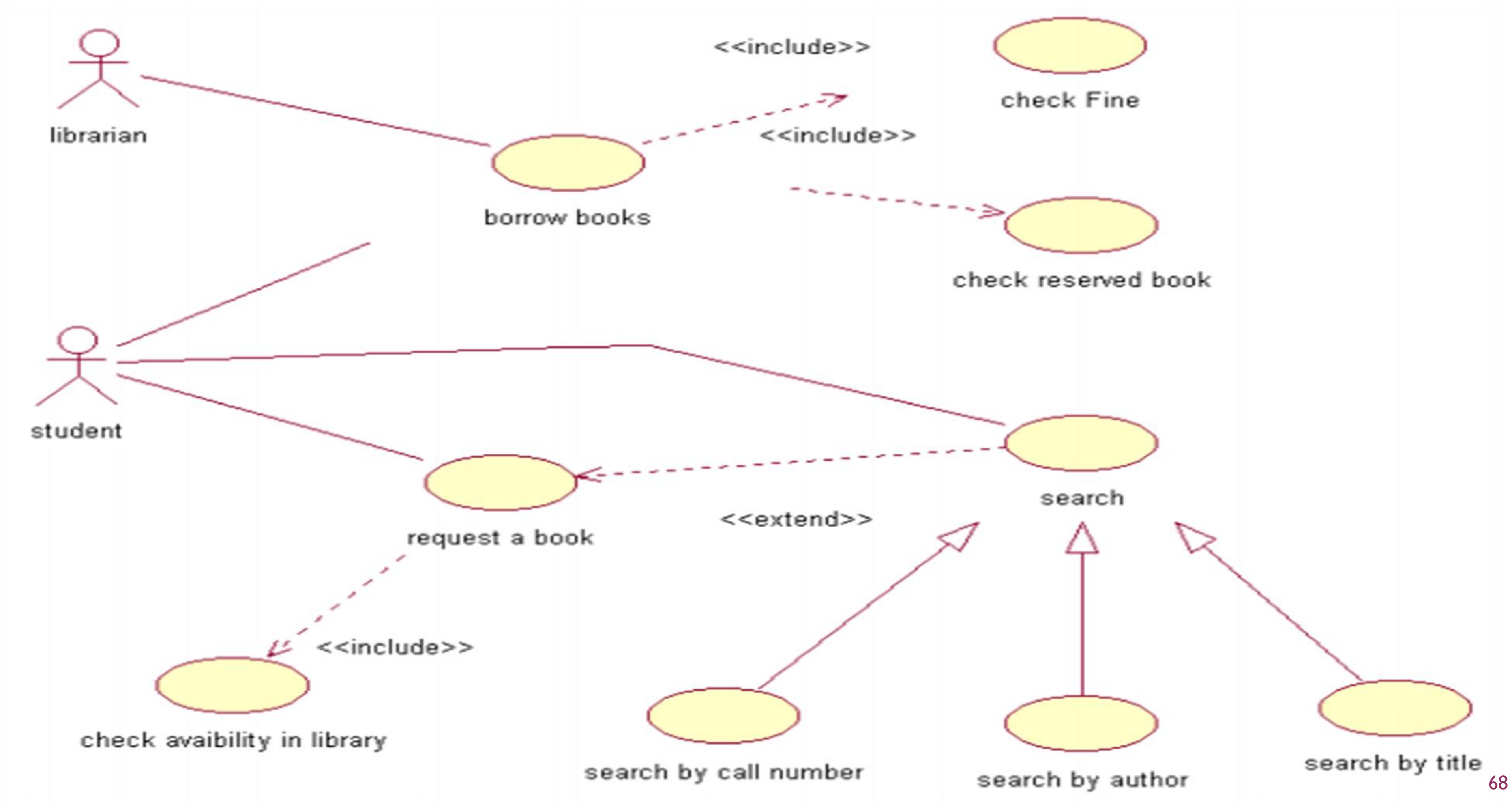


EXAMPLE – INCLUDE AND EXTEND

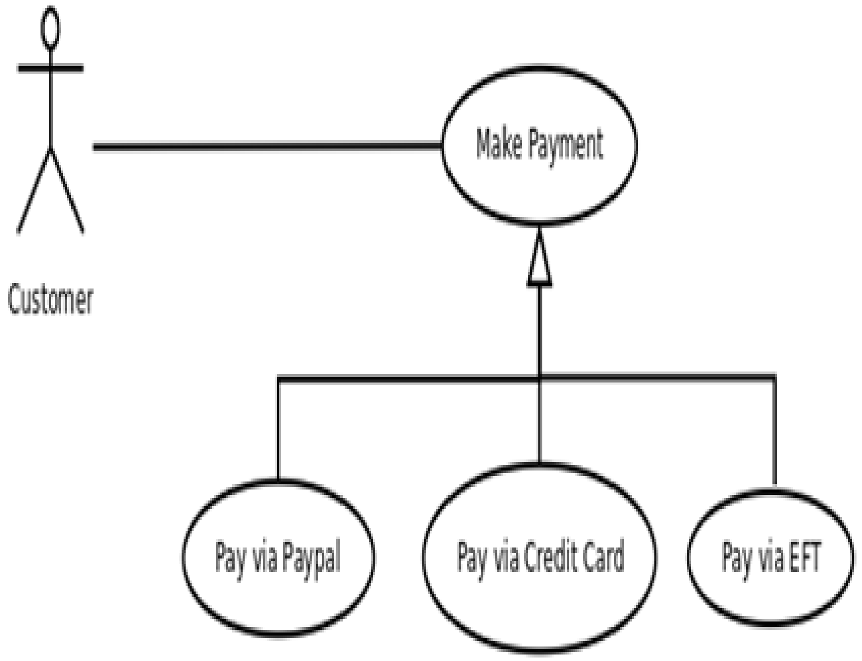


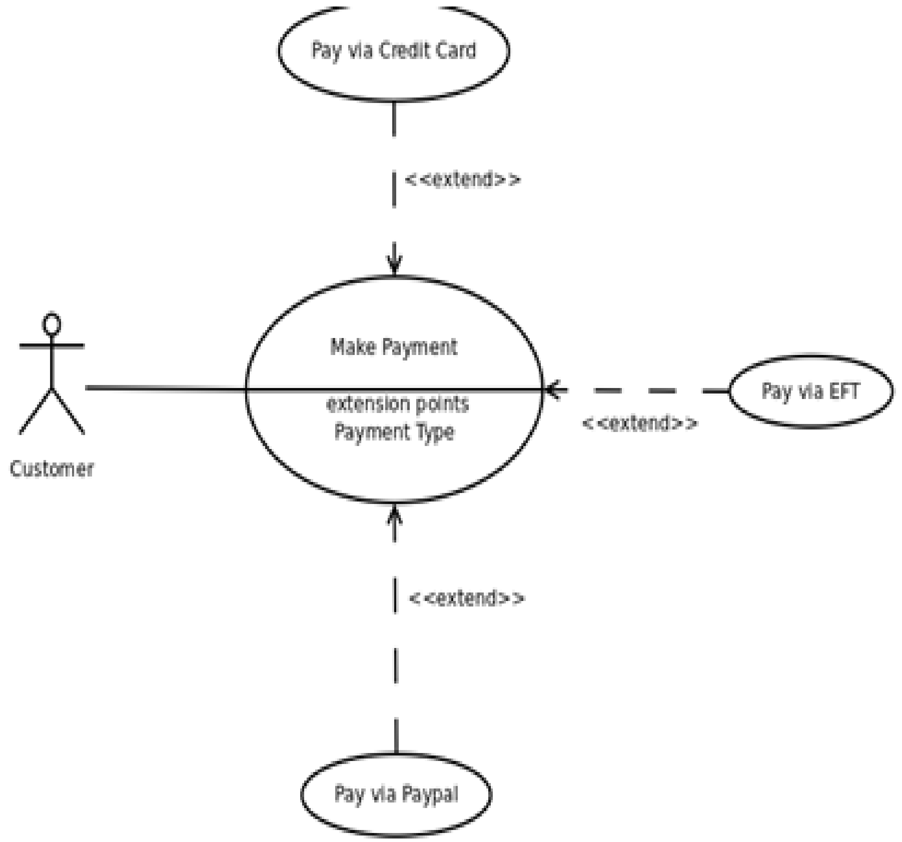
The use of include and extend is discouraged simply because they add unnecessary complexity to a Use Case diagram.

Difference between Generalization and Extend



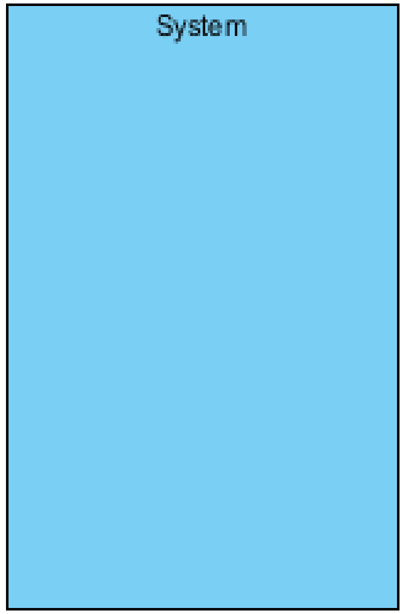
Difference between Generalization and Extend



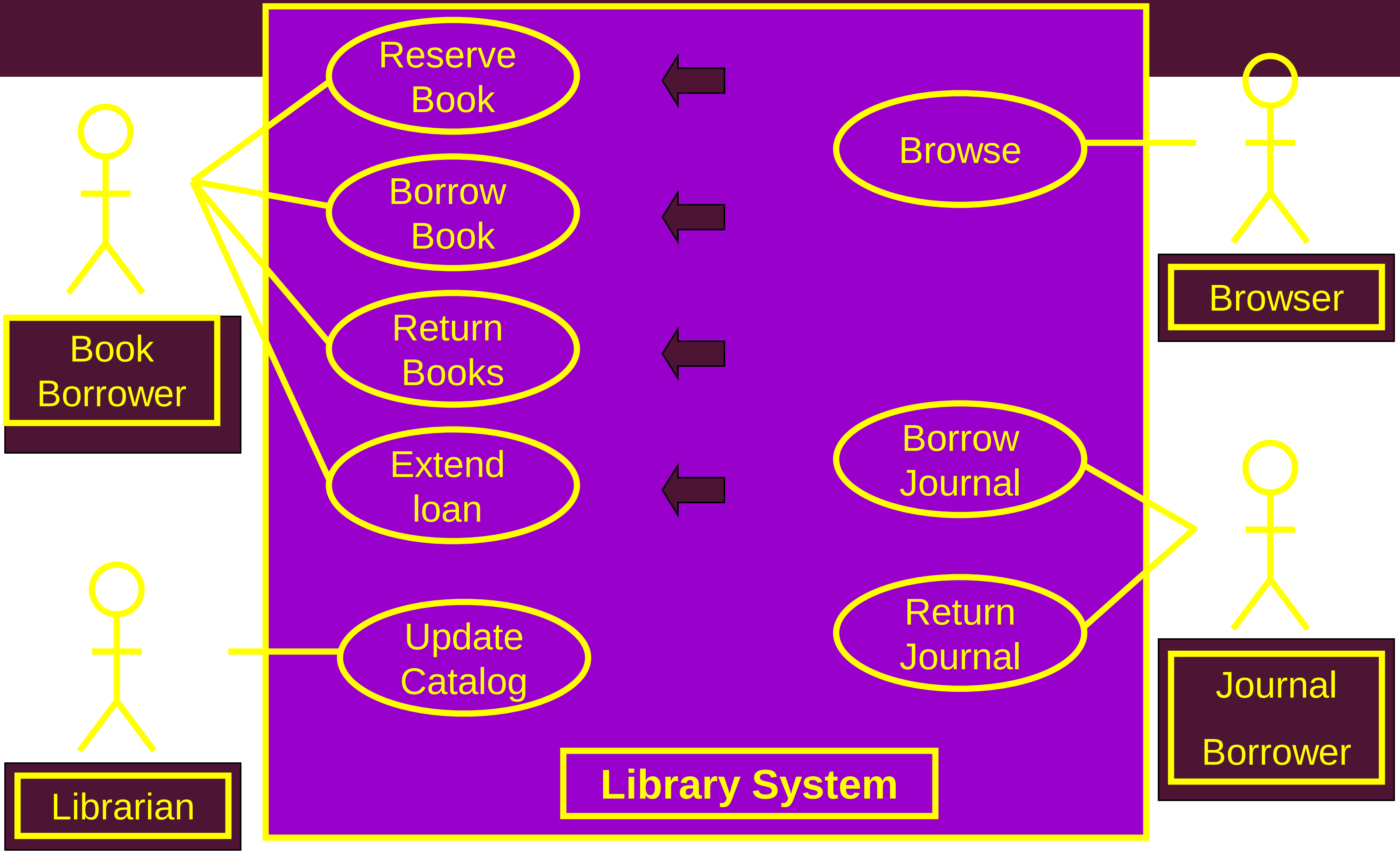


USE CASE - BOUNDARY

- Boundary
  - A boundary rectangle is placed around the perimeter of the system to show how the actors communicate with the system.
  - A system boundary of a use case diagram defines the limits of the system.



USE DIAGRAM FOR A LIBRARY SYSTEM



TYPES OF USE CASES

- **Brief:**
- Terse one-paragraph summary, usually of the main success scenario.
- Used during early requirements analysis, to get a quick sense of subject and scope. May take only a few minutes to create.
- **Example:**
- **Process Sale:** A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.

TYPES OF USE CASES

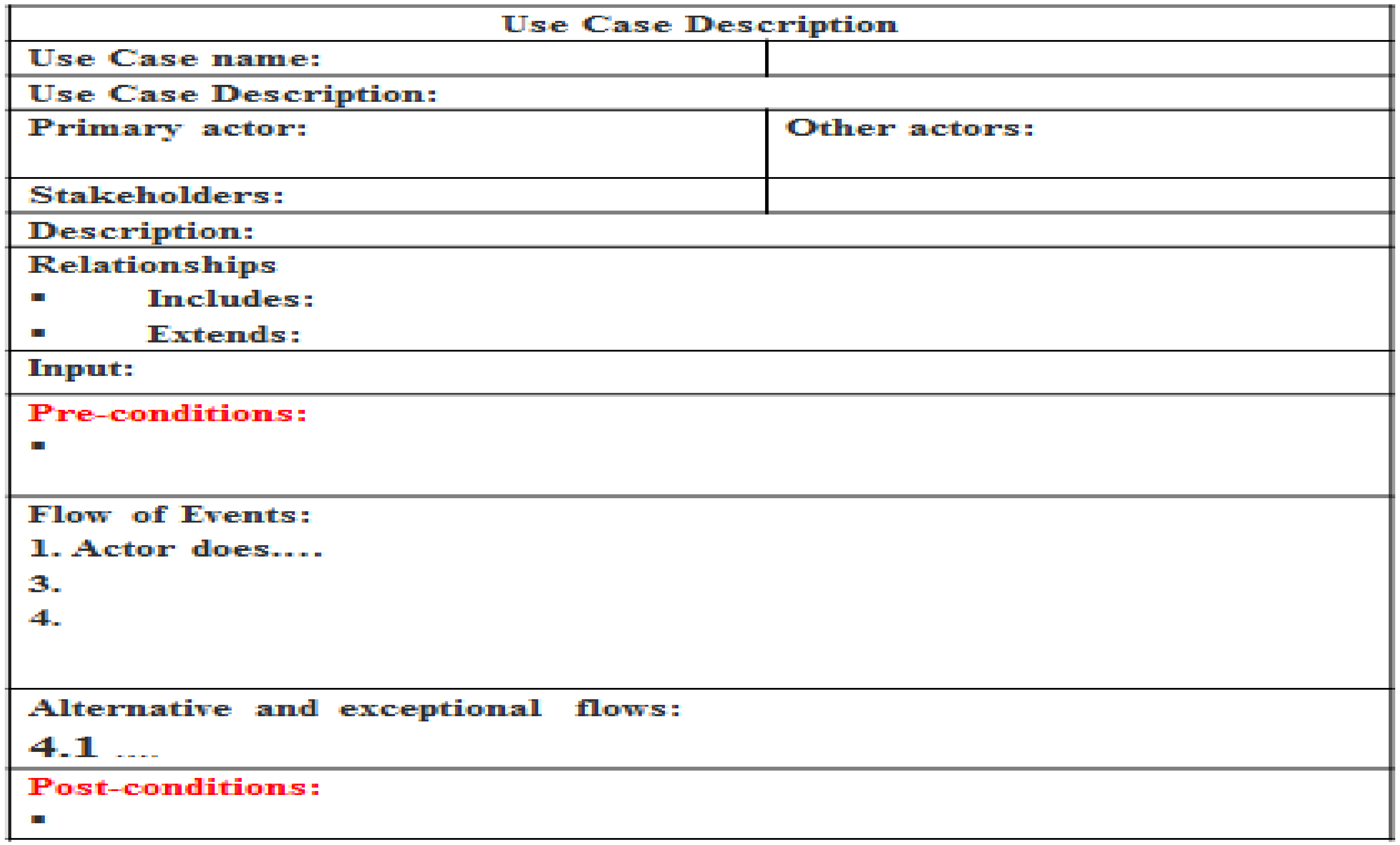
- <u>**Casual**</u>
- Informal paragraph format. Multiple paragraphs that cover various scenarios.
- Used during early requirements analysis, to get a quick sense of subject and scope. May take only a few minutes to create.
- <u>**Example:**</u>
- <u>**Handle Returns:**</u>
- Main Success Scenario: A customer arrives at a checkout with items to return. The cashier uses the POS system to record each returned item …
- Alternate Scenarios: If the customer paid by credit, and the reimbursement transaction to their credit account is rejected, inform the customer and pay them with cash. If the item identifier is not found in the system, notify the Cashier and suggest manual entry of the identifier code (perhaps it is corrupted).

TYPES OF USE CASES

- **Fully dressed**
- All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.
- After many use cases have been identified and written in a brief format, then during the first requirements workshop a few (such as 10%) of the architecturally significant and high-value use cases are written in detail.

A RECOMMENDED TEMPLATE

FULL USE CASE DESCRIPTION



Use Case Description	
Use Case name:	
Use Case Description:	
Primary actor:	**Other actors:**
Stakeholders:	
Description:	
Relationships ▪ **Includes:** ▪ **Extends:**	
Input:	
Pre-conditions: ▪	
Flow of Events: **1. Actor does....** **3.** **4.**	
Alternative and exceptional flows: **4.1 ....**	
Post-conditions: ▪	

HOW TO FIND USE CASES?

- 1. Choose the system boundary. Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?
- 2. Identify the primary actors. Those that have goals fulfilled through using services of the system.
- 3. Identify the goals for each primary actor.
- 4. Define use cases that satisfy user goals; name them according to their goal. Usually, user-goal level use cases will be one-to-one with user goals, but there is at least one exception, as will be examined.

USE CASES: HOW BIG OR SMALL?

- It really depends on context
- Generally, the use case should define a task performed by one actor in one plat one time, in response to an event.
- If the use case is growing to many pages in length, consider creating sub-tasIf the use case is only one sentence or step, probably too small and not worth
- exploring
- Is there a basic business value in the use case?
- Which actors are impacted and how?
- What is the impact to the overall business?

USE CASES

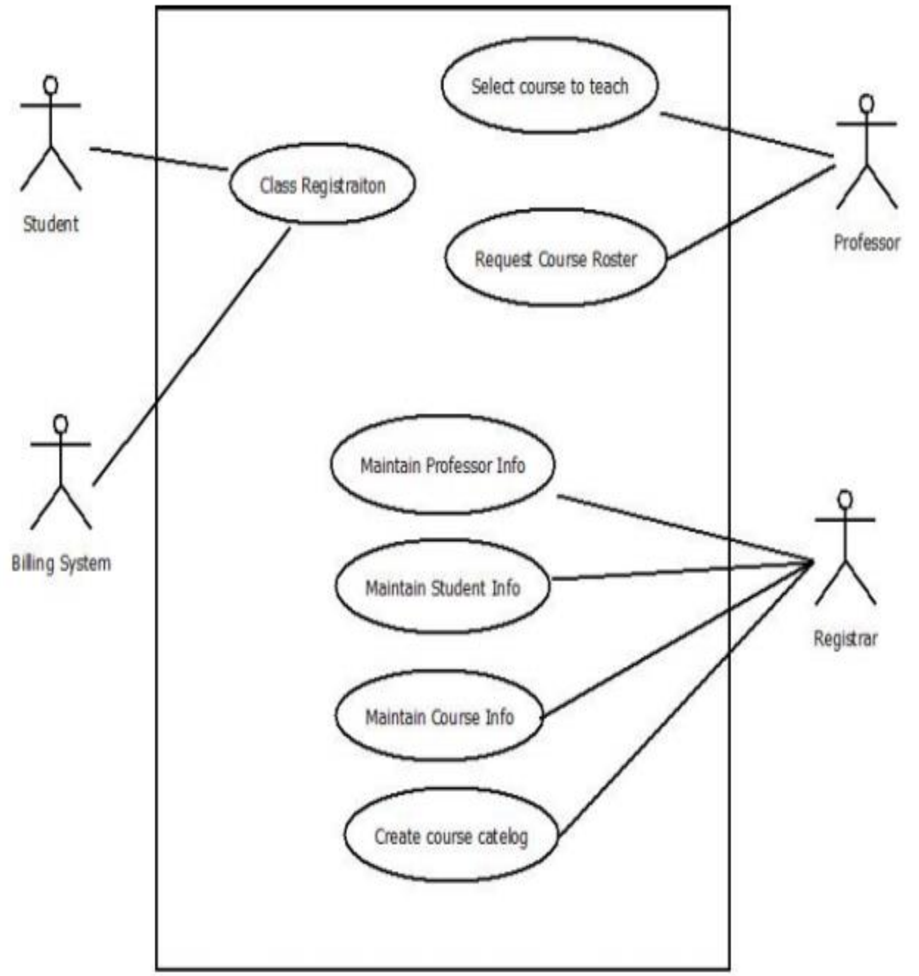
- Here is a scenario for a purchasing items.

- “Customer arrives at checkout with items to purchase in cash. Cashier records the items and takes cash payment. On completion, customer leaves with items. "

- We want to write a use case for this scenario.
- Remember: A **use case** is a summary of scenarios for a single task or goal.

YOUR TURN: ALTERED STATE UNIVERSITY (ASU) REGISTRATION SYSTEM

- Professors indicate which courses they will teach on-line.
- A course catalog can be printed which is created by Registrar.
- Allow students to select on-line courses for upcoming semester.
- When the registration is completed, the system sends information to the billing system.
- Professors can obtain course rosters on-line.
- Students can add or drop classes on-line.
- Registrar maintains all the information about student, course and professor.



CASE STUDY – WRITE A USE CASE

- An executive at a chain of family restaurants called Brampton's Pizza and Pasta. They are looking to develop some software for in-house use at their restaurants. Their goal is to take the restaurant ordering process and make it more efficient. She is thinking that customers should be able to view the menu of the restaurant they're in, and once they're ready, place an order. She'd really like there to be a kids' page where you can see the kids' menu. Maybe there's a few games for the kids to play, but most importantly, it should be easy enough to use that kids can make an order themselves. Customers should also be able to specify any changes they'd like to make for their meal, and they should be able to list any dietary restrictions they may have before they submit their order to the kitchen. The kitchen should then be able to view these orders as they come in. Customers should be able to view and pay their bill within the system.

IDENTIFY USE CASES

- **Automated Teller Machine (ATM)** is a banking system.
- This banking system allows customers or users to have access to financial transactions. These transactions can be done in public space without any need for a clerk, cashier, or bank teller.
- The user is authenticated when enters the plastic ATM card in a Bank ATM, Then enters the user name and PIN (Personal Identification Number). For every ATM transaction, a Customer Authentication is required and essential.
- User checks the bank balance as well as also demands the mini statement about the bank balance if they want. Then the user withdraws the money as per their need. If they want to deposit some money, they can do it. After complete action, the user closes the session. If there is any error or repair needed in Bank ATM, it is done by an ATM technician. ATM technician is responsible for the maintenance of the Bank ATM, upgrades for hardware, firmware or software, and on-site diagnosis.

FUNCTIONAL REQUIREMENTS

The Problem:

Currently, there are no applications dedicated to students in universities. Students want to participate in university courses and see announcements in these courses, but have this information spread out on multiple websites, which are hard to find. They want to find fellow students with the same interests and share their opinion about courses and course material, however this is not possible yet. They also want to discuss exam questions and find the right place where the exam takes place without searching on Google.

FUNCTIONAL REQUIREMENTS

Scenarios:

Jack, an incoming student from US, is studying computer science at TUM. He has business administration as minor subject and is already used to visit the courses in the computer science building. The business administration courses however, are located in a lecture hall in another building in the city center of Munich. He never visited the other building before, so he does not know how to find the lecture halls for his minor subject. He browses through the courses in the course catalog of the University App and finds the course "Foundations of Business Administration" with course times and the location of the lecture hall on a map. While he is attending the course, he makes contact with fellow students who also attend the course and reads their comments. He likes one comment "Great exercises" by Jenny, who is also studying informatics. From Jenny's picture, he remembers that they met a week ago at the coffee machine. He requests friendship with Jenny (she might help him to pass the final) and adds a new comment about exam questions from earlier exams. While he is browsing, Jenny is notified about the friend request and accepts it. jack, in turn, is notified that Jenny has accepted his request and now browses through all the courses that Jenny is visiting. He finds another interesting course "Cost Accounting" that he wants to visit and saves it into his course list.

POSSIBLE SOLUTIONS

- FR1: Search for available courses: A student can see all courses of the current semester in his major and minor subject. He is able to join the course which saves it into his course list. He can also drop a course.
- FR2: Check course details: A student can see details about a course such as the course times, the location of the lecture hall on a map and other course attendees including their name and picture.
- FR3: Update profile: A student can update his profile settings and his profile picture. He can also change the notification settings.
- FR4: Add comments: A student can add comments about a course and thus start a discussion. Others can like the comment and write follow-up comments.
- FR5: Request friendship: A student can request friendship with another student who then receives a notification about the request. The second student can accept and reject friendship which both notifies the first student.
- FR6: Browse friends' courses: A student can browse the courses of his friends.
- FR7: View announcements: A student can view course announcements and comment/like them.
- FR8: Post updates to timeline: A student can post updates to his timeline. Friends are notified about updates and can comment and like them. Certain updates are posted automatically such as saving a course into the course list or commenting on a course.
- FR9: See course calendar: A student can see all courses in a calendar.

That is all