INTRODUCTION TO SOFTWARE ENGINEERING

Software Engineering (SE)

- × The term software engineering was popularized during the 1968 NATO Software Engineering Conference (held in Garmisch, Germany) by its chairman F.L. Bauer, and has been in widespread use since.
- × The field of SE aims to find answers to the many problems that software development projects are likely to meet when constructing large software systems.
- × Such systems are complex
  - because of their sheer size,
  - because they are developed by a team involving people from different disciplines, and
  - because they will be modified regularly to meet changing requirements, both during development and after installation.
- × The software developed is according to:
  - → Accepted industry practice, with good quality control, adherence to standards, and in an efficient and timely manner.
- × The term **software engineering** refers to a systematic procedure that is used in the context of a generally accepted set of goals for the analysis, design, implementation, testing and maintenance of software.
- × According to IEEE **SOFTWARE ENGINEERING** is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.
- × The discipline of software engineering encompasses knowledge, tools, and methods for defining software requirements, and performing software design, software construction, software testing, and software maintenance tasks.
- × **SOFTWARE ENGINEERING** also draws on knowledge from fields such as computer engineering, computer science, management, mathematics, project management, quality management, software ergonomics, and systems engineering. This is according to the author Ronald Leach.
- × The software produced should be efficient, reliable, usable, modifiable, portable, testable, reusable, maintainable, interoperable, and correct.
- × SWEBOOK (Software Engineering Body of Knowledge), 2013 defines these characteristics as the following
- × The software produced should be:
  - **Efficient** – software made in expected time and within resources
  - **Reliable** – software performs as expected
  - **Usable** – software can be used properly
  - **Modifiable** – software can be easily changed as requirement changes
  - **Portable** – system can be ported to other computers without major rewriting of the software
  - **Testable** – software can be easily tested
  - **Reusable** – software can be used again in other projects
  - **Maintainable** – software can be easily understood and changed overtime if problems occur
  - **Interoperable** – software can interact with other systems properly
  - **Correct** – the program produces the correct output

Where Does The Software Engineer Fit In?

Software Engineering:

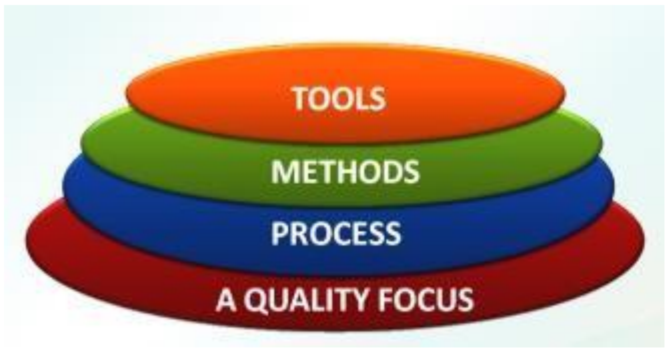
- focusing on computer as a problem-solving tool
- Relationship between computer science and software engineering.

But the question is... why is there a need for software engineering?

Over the past years

- Technology has advanced rapidly
  - Ordering of food, clothes, or even transportation is done through smartphones, while on the phone and en route to your next destination
- With this example, software has grown to provide humanity with endless opportunities
- More complex is the system, the more is the programming skill needed.
- In a nutshell, software engineering is developing complex software end products by professionals within time and budget.
- Software engineering is often viewed as layered.

SOFTWARE ENGINEERING IS A LAYERED TECHNOLOGY AS FOLLOWS:



SOFTWARE ENGINEERING LAYERS

Quality Focus

→ The bedrock that supports software engineering. Any engineering approach must rest on an organizational commitment to quality.

Process

→ It is the glue that holds the technology layers together and enables rational and timely developments of computer SW.

→ Process defines a framework that must be established for effective delivery of SW engineering technology.

→ It forms the basis for

- management control of software projects
- establishes the context (technical methods applied)
- Work products (models, documents, data, reports, form, etc.) are produced,
- milestones are established,
- quality is ensured, and change is properly managed

Methods

→ Provide the technical how-to's forbuilding SW.

→ Encompasses a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.

→ Rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Tools

→ Provide automated and semi automated support for the process and the methods.

→ When tools are integrated so that information created by one tools can be used by another, a system for the support of SW development, called CASE (Computer-Aided Software Engineering), is established.

CHALLENGES IN DEVELOPING SOFTWARE

1. Developing / large / complex software application
2. Effort intensive
3. High cost (in case the solution is unsuccessful)
4. Long development time
5. Changing needs for users
6. High risk of failure, reasons of user acceptance, performance and maintainability

What is Good Software? Perspective on Quality

Dr. David A. Garvin, PhD, a professor of Business Administration Harvard Business school identified 5 major approaches of defining quality

- The **transcendent** view
- The **product-based** view
- The **user-based** view
- The **manufacturing-based** view
- The **value-based** view

QUALITY: Transcendent

Transcendent

› Quality is an innate excellence (cannot be defined exactly
› Quality is simple, unanalyzable recognized only through experience

Product-Based

› Quality is precise, measurable variable in the components and attributes of a product
› Reflects the presence or absence of such measurable and desired product attributes

QUALITY: User-Based

User-based

› Goods that best satisfies user preferences have the highest quality

QUALITY: Manufacturing-Based

Manufacturing based

› Conformance to requirements
› Preventing defects (less expensive than repair/rework)
› Products that follows specification are of high quality

QUALITY: Value-based

Value Based

› Provides quality product at an acceptable price or conformance at an acceptable cost
› This approach is becoming the more prevalent

What is Good Software?

× Good software engineering must always include a strategy for producing quality software.
× Three ways of considering quality
  - The quality of the product
  - The quality of the process
  - The quality of the product in the context of the business environment

The Quality of the Product

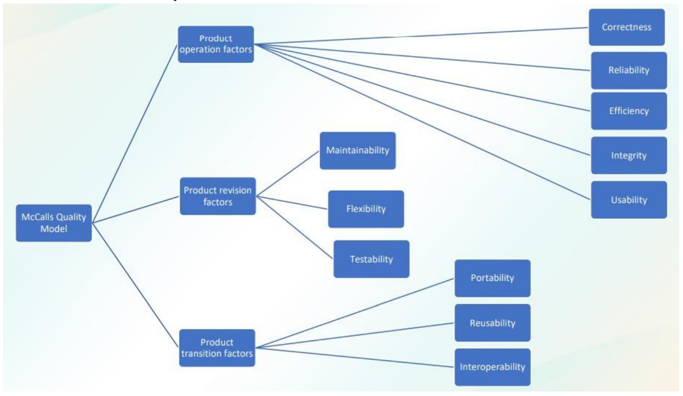
✓ Users judge external characteristics (e.g., correct functionality, number of failures, type of failures)
✓ Designers and maintainers judge internal characteristics (e.g., types of faults)
✓ Different stakeholders may have different criteria
✓ Need quality models to relate the user's external view to developer's internal view
✓ Classic model of software quality factors was suggested by McCall (McCall et al., 1977)
✓ McCall's factors consists of 11 software quality factors

- ✓ Similarly models suggested by Deutsch and Willis (1988) and Evans and Marciniak (1987)
- ✓ All these models do not differ substantially from McCall's model
- ✓ McCall factor model provides a practical, up-to-date method for classifying software requirements (Pressman, 2000).

McCall's Quality Model

1. Correctness
2. Reliability
3. Efficiency
4. Integrity
5. Usability
6. Maintainability
7. Flexibility
8. Testability
9. Portability
10. Reusability
11. Interoperability

- ✓ The model was grouped into three categories (product operation factors, product revision factors, product transition factors)



- ✓ These qualities are also subdivided based on the external and internal qualities
- ✓ External quality is the quality of the finished product, the quality as it appears to the external world, as it comes of the end of the assembly line. These are factors that are clearly visible to end users.
- ✓ Internal quality is the quality of the product as it is being constructed, while it is on the assembly line. It is concerned with internal technical issues of the software.
- ✓ In general, users of the software only care about external qualities, but it is the internal qualities – which deal largely with the structure of the software – that help developers achieve external qualities.

External Quality	Integrity Reliability Usability Correctness
Internal Quality	Efficiency Maintainability Testability Flexibility Interoperability Reusability Portability

The Quality of the Process

- ✓ Quality of the development and maintenance process is as important as the product quality
- ✓ The development process needs to be modeled
- ✓ Modeling will address questions such as
  - › Where to find a particular kind of fault
  - › How to find faults earlier
  - › How to build in fault tolerance
  - › What are alternative activities
- ✓ This is similar to the process called software quality management (SQM)
- ✓ SQM is a management process that aims to develop and manage the quality of the software in such a way so to best ensure that the product meets the quality standards expected by the customer while also meeting any necessary regulatory and developer requirements is any

- ✓ The following are the Quality management activities that can ensure the quality of the software
  1. Quality assurance
  2. Quality planning
  3. Quality control
- ✓ The following are the Quality management activities that can ensure the quality of the software
  - × **Quality assurance**
    - sets up an organized and logical set of organizational processes and deciding on that software development standards
    - industry best practices paired with those organizational processes, software developers stand a better chance of producing higher quality software.
  - × **Quality planning**
    - Quality planning works at a more granular, project-based level,
    - defining the quality attributes to be associated with the output of the project, and
    - how those attributes should be assessed.
  - × **Quality control**
    - The quality control team tests and reviews software at its various stages to ensure quality assurance processes and standards at both the organizational and project level are being followed.

The Quality in the Context of the Business Environment

- ✓ Models for process improvement
  - › SEI's Capability Maturity Model (CMM)
  - › ISO 9000
  - › Software Process Improvement and Capability dEtermination (SPICE)
- ✓ Business value is as important as technical value
- ✓ Business value (in relationship to technical value) must be quantified
- ✓ A common approach: **return on investment (ROI)**
  - › what is given up for other purposes
- ✓ ROI is interpreted in different terms: reducing costs, predicting savings, improving productivity, and costs (efforts and resources)

REASONS FOR SOFTWARE FAILURE

- schedule slippage (software is not really time)
- cost overruns (the software was abandoned in the middle because of the high costing requirement)
- does not solve user's problem (not useful)
- poor quality
- poor maintainability

What Leads To Such Problems

- No planning of development work
- Deliverables to user not identified
- Poor understanding of user requirement
- No control of review (review and control systematically; cost time and resources)
- Technical incompetence (the need to understand the changes in technologies, new tools and techniques in recent trends in technologies)
- Poor understanding of cost and effort both from user and the developer.

CHALLENGES FACING SOFTWARE ENGINEERING

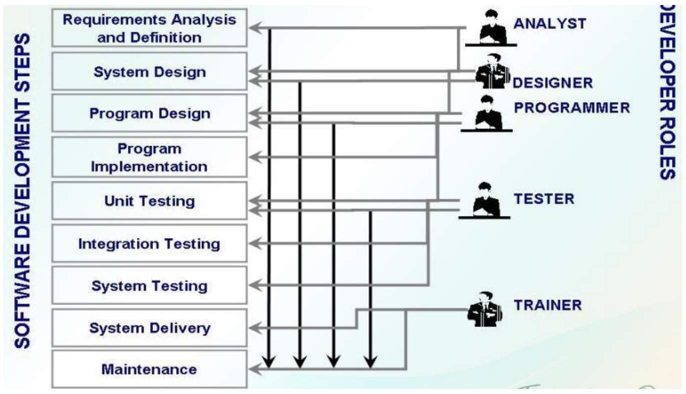
- Legacy challenge.
- Heterogeneity challenge.
- Delivery challenge.

Who Does Software Engineering?

- **Customer**: the company, organization, or person who pays for the software system
- **Developer**: the company, organization, or person who is building the software system
- **User**: the person or people who will actually use the system

MEMBERS OF THE DEVELOPMENT TEAM

- **Requirement Analysts** – work with the customers to identify and document the requirements
- **Designers** - generate a system-level description of what the system is suppose to do
- **Programmers** - write lines of code to implement the design
- **Testers** - catch faults
- **Trainers** - show users how to use the system
- **Maintenance Team** - fix faults that show up later
- **Librarians** - prepare and store documents such as software requirements
- **Configuration Management Team** - maintain correspondence among various artifacts
- Typical roles played by the members of a development team



SOFTWARE DESIGN

Software Architecture

× Software architecture is commonly used to describe the organization of software systems.

× An <u>architecture</u> is the definition of the key elements that constitute a system, their relationships, and any constraints.

× Generally speaking, architectures of software-based systems can be classified into several categories.

Software Architectures Categories:

Business Architecture

→ Described the structure of the business tasks performed by the organization

→ These are created by business analysts

Physical Architecture

→ Describes the structure of the hardware platform(s) or network(s) on which the system will operate

→ Physical architecture of a system

Logical Architecture

→ Describes the structure of the business and application objects

→ Is often part of an object-oriented view ofa system

→ Is often created by an object analyst

Functional Architecture

→ Describes the structure of the potential use cases and the requirements from a user's point of view

→ Is often part of an object-oriented view of a system

Software Architecture

→ Describe the structure of the system into layers, such as OSI (Open systems interconnection) or layered architecture of UNIX operating system.

→ Decomposed to client and server

Technical Architecture

→ Describes the structure of the major interfaces in the software architecture

→ Elements include:

- Application programming interfaces (APIs)
- Middleware
- Database management systems
- Graphical User Interfaces
- Other glueware or bridgeware needed to interface components

→ Decisions to use COBRA, DCOM, Java RMI, or RPC or even cloud computing, with a protocol such as HTTP, SOAP, REST, or JSON, would be reflected in this type of system architecture

System Architecture

→ Describes the structure of the business, application, and technical objects and their relationships.

→ The description is often created by a systems analysts

Deployment Architecture

→ Describes the structure of the mapping of the system and technical architectures onto the physical architecture

→ It includes two views:

- A static view of the basic files that will be necessary
- A dynamic view of the concurrent processes and threads, and the mechanisms for their synchronization and communication.

→ A thread on a particular computer or to have an autonomous

Deployment Architecture

→ A thread on a particular computer or to have an autonomous agent performing computation would be reflected in thistype of architecture.

Software Architecture Importance

- Software architecture are an enabler for communication between stakeholders
- The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows

ARCHITECTURAL DESIGN

- The software must be placed into context
  - The design should define the external entities (other systems, devices, people) that the software interacts with and the nature ofthe interaction
  - A set of architectural archetypes should be identified
  - An archetype is an abstraction (similar to a class) that represents one element of system behavior.
  - The designer specifies the structure of the system by defining and refining software components that implement each archetype

DATA DESIGN (Data Modeling Concepts)

At the architectural level…

- Design databases to support the application architecture
- Design of methods for 'mining' the content of multiple databases
- Navigate through existing databases to extract information.
- Design of a data warehouse

ENTITY RELATIONSHIP DIAGRAM

Entity-Relationship Diagram (ERD) represents all data objects entered, stored, transformed,and produced within an application

Entity or Entity Types

- An ERD is consists of entities or data objects, data attributes and relationships with the entities.
- An **entity** or an **entity type** or a **data object** is a representation of something that has several different properties (attributes) that must be understood by software. It is anything that produce or consumes information.
- Entities:
  - **Entity instance** – person, place, object, event, concept (often corresponds to a row in a table)
  - **Entity Type** – collection of entities (often corresponds to a table)
- Example of entity or entity types
  - **Attribute** – property or characteristic of an entity or relationship type (often corresponds to a field in a table)

Relationships

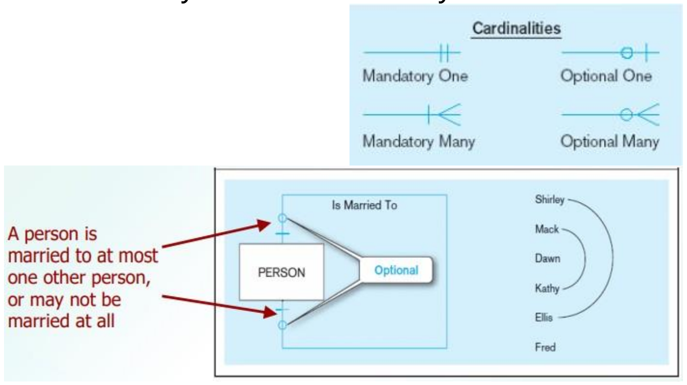
- **Relationship instance** – link between entities (corresponds to primary key-foreign key equivalencies in related tables)
- **Relationship Type** – category of relationship…link between entity types

RELATIONSHIPS

- Relationships can have attributes
  - Two entities can have more than one type of relationship between them (multiple relationships)
  - **Associative Entity** – combination of relationship and entity

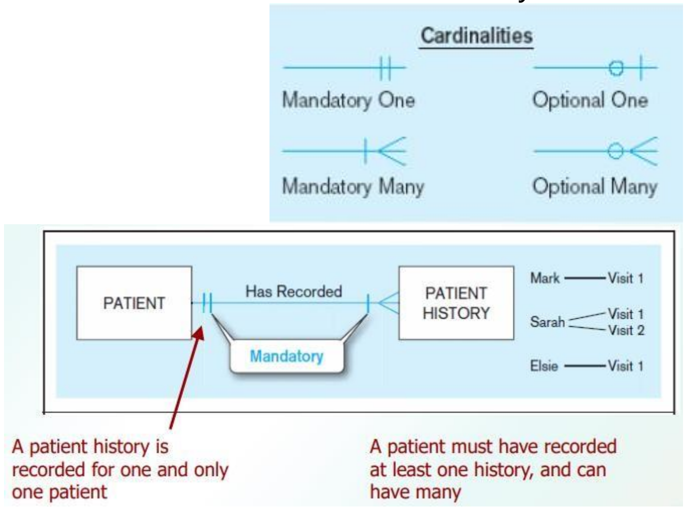
One-to-One (1:1)

→ Each entity in the relationship will have exactly one related entity



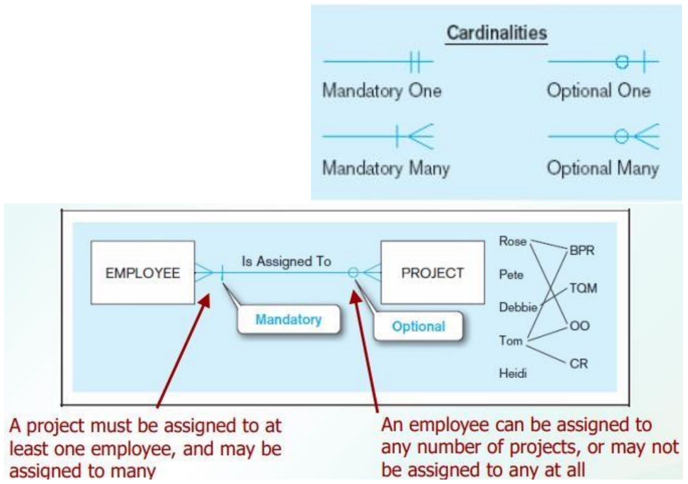
One-to-Many (1:M) or Many-to-One (M:1)

→ An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity.



Many-to-Many (M:M)

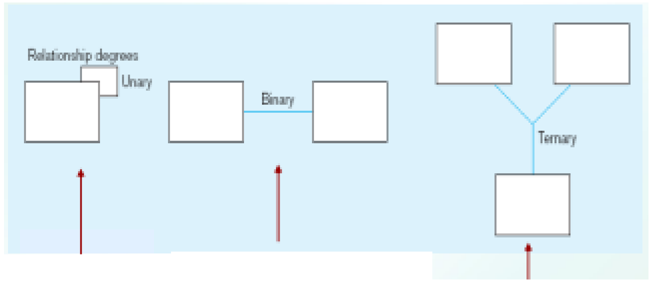
→ Entities on both sides of the relationship can have many related entities on the other side


Cardinalities
Mandatory One
Optional One
Mandatory Many
Optional Many
EMPLOYEE
Is Assigned To
PROJECT
Mandatory
Optional
Rose
Pete
Debbie
Tom
Heidi
BPR
TQM
OO
CR
A project must be assigned to at least one employee, and may be assigned to many
An employee can be assigned to any number of projects, or may not be assigned to any at all

RELATIONSHIPS

× Degree of a relationship is the number of entity types that participate in it.
- Unary Relationship
- Binary Relationship
- Ternary Relationship

Unary Relationship

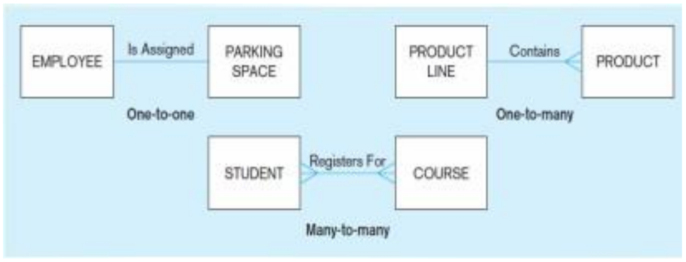


Oneentityrela

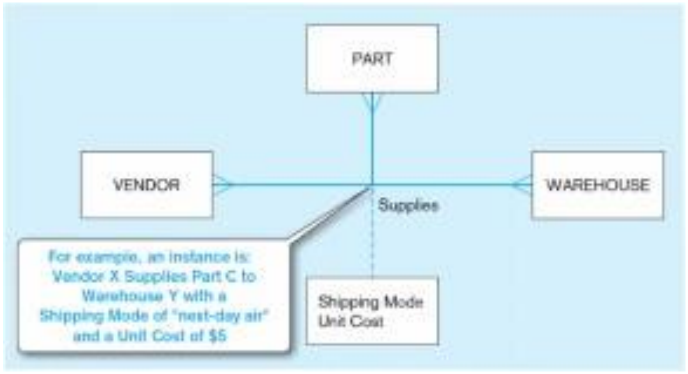
y
t
y
p
e

Entities of

Binary Relationship



Ternary Relationship


PART
VENDOR
WAREHOUSE
Supplies
For example, an instance is: Vendor X Supplies Part C to Warehouse Y with a Shipping Mode of "next-day air" and a Unit Cost of $5
Shipping Mode
Unit Cost

STRUCTURED DESIGN

× A mapping technique, called **structured design** is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram to software architecture.

× The transition from information flow (represented as a DFD) to program structure is accomplished as part of a six step process.

× These six step processes are:

1) the type of information flow is established,
2) flow boundaries are indicated,
3) the DFD is mapped into the program structure,
4) control hierarchy is defined,
5) the resultant structure is refined using design measures and heuristics, and
6) the architectural description is refined and elaborated.

Data Flow Diagram

× A **Data Flow Diagram (DFD)** is a graphical representation of the "flow" of data through an information modeling its <u>process</u> aspects.

× A DFD shows what kinds of information will be input to and output from the system, where the data will come from and go to,and where the data will be stored.

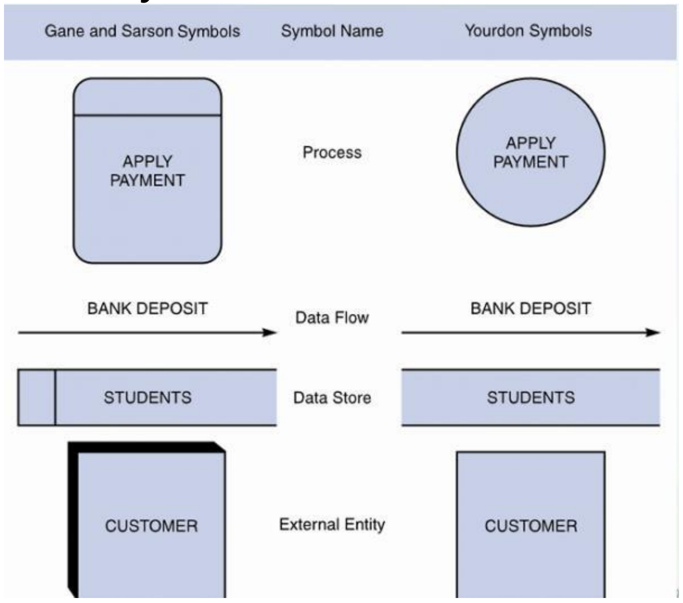
× The DFD takes an input-process-output view of a system. Data objects are represented by labeled arrows, and

× Transformations are represented by circles (also called bubbles).

× The DFD is presented in a hierarchical fashion.

× Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequent level.

DFD Symbols



Creating a Set of DFD

Here are simple guidelines

1. the level 0 data flow diagram should depict the software/system as a single bubble;
2. primary input and output should be carefully noted;

3. Refinement should begin by isolating candidate processes, data objects, and data stores to be represented at the next level;
4. all arrows and bubbles should be labeled with meaningful names;
5. information flow continuity must be maintained from level to level, and
6. one bubble at a time should be refined

Before we proceed to diagram 0, lets discuss what SafeHome does

× The SafeHome security function enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through the Internet, a PC, or a control panel.

× During installation, the SafeHome PC is used to program and configure the system.

× Each sensor is assigned a number and type,a master password is programmed for arming and disarming the system, and telephone number(s) are input for dialing when a sensor event occurs.

× When a sensor event is recognized, the software invokes an audible alarm attached to the system. After a delay time that isspecified by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, provides information about the location, reporting the nature of the event that has been detected.

× The telephone number will be redialed every 20 seconds until telephone connection is obtained.

× The homeowner receives security information via a control panel, the PC, or a browser, collectively called an interface. The interface displays prompting messages and system status information on the control panel, the PC, or the browser window. Homeowner interaction takes the following form...

UNIFIED MODELING LANGUAGE

× Unified Modeling Language (UML) is "a standard language for writing blueprints. UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system"

× Just as building architects create blueprints

× Software architects create UML diagrams

Two main types of diagrams in the UML

1. **Structural Diagram** – are used to describe the relationship between classes
2. **Behavioral Diagram** – can be used to describe the interaction between people and the thing we refer to as a use case, or how the actors use the system.

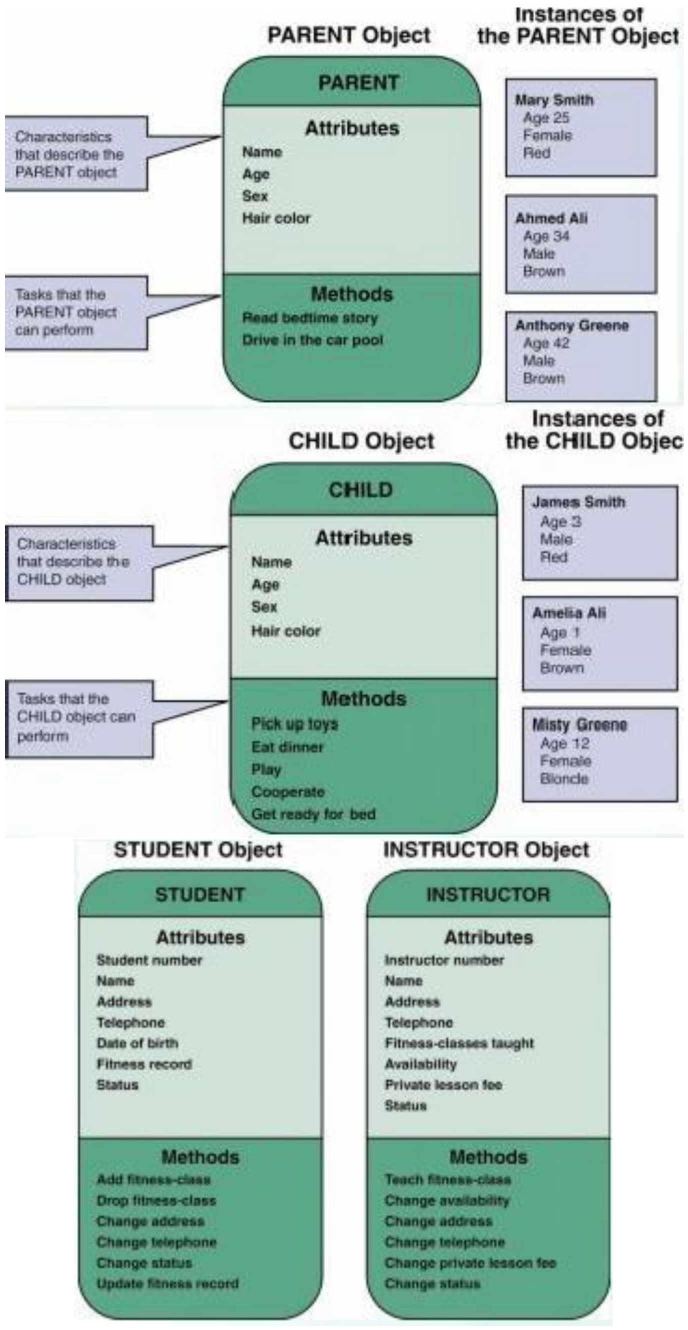
× It all begins with the construction of a model.

× A **model** is an abstraction of the underlying problem. The **domain** is the actual world from which the problem comes.

× Models consist of **objects** that interact by sending each other **messages**. Think of an object as "alive." Objects have things they know (**attributes**) and things they can do (**behaviors** or **operations**). The values of an object's attributes determine its **state**.

× **Classes** are the "blueprints" for objects. A class wraps attributes (data) and behaviors (methods or functions) into a single distinct entity. Objects are **instances** of classes.

OBJECTS CONCEPTS



Attributes

- × If objects are similar to nouns, attributes are similar to adjectives that describe the characteristics of an object
- × Some objects might have a few attributes; others might have dozens state

Methods

- × A method defines specific tasks that anobject can perform
- × Just as objects are similar to nouns and attributes are similar to adjectives, methods resemble verbs that describe what and howan object does something

USE CASE

USE CASE DIAGRAM

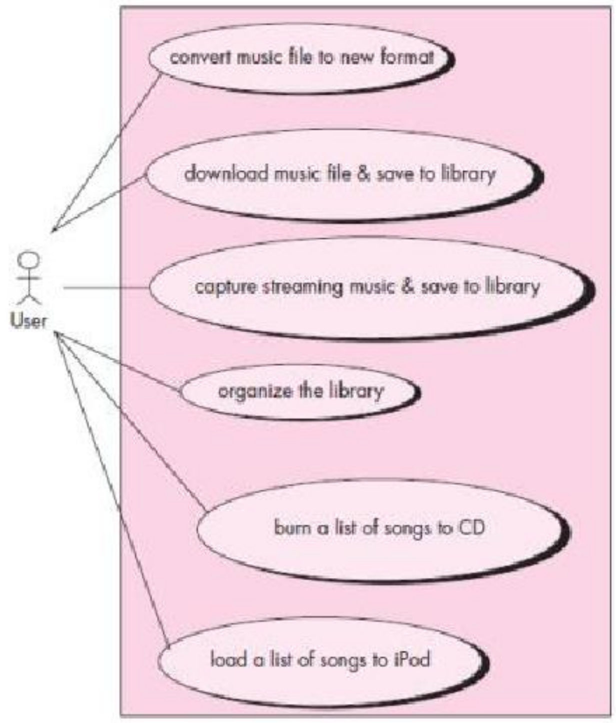
- × **Use case diagrams** describe what a system does from the standpoint of an external observer. The emphasis is on what a systemdoes rather than how.
- × Use case diagrams are closely connected to scenarios. A **scenario** is an example of what happens when someone interacts with the system. Here is a scenario for a managing user digital files.

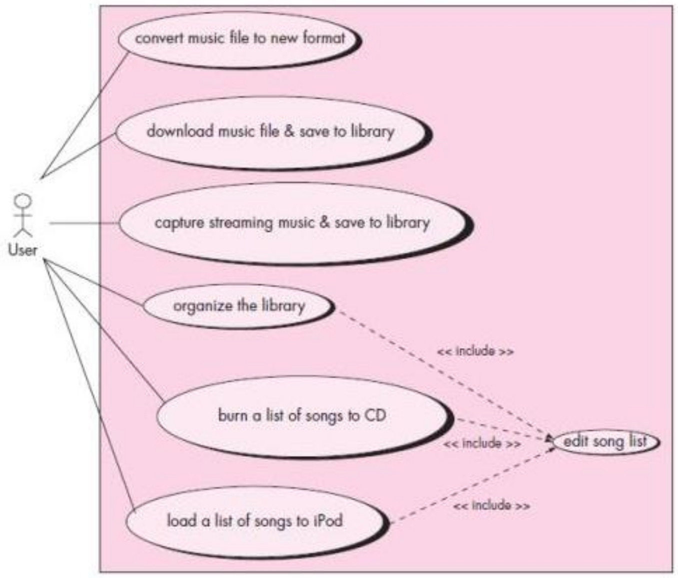
Use Case Diagram Example

- × Let us say, there is a software application for managing digital music files, similar to Apple's iTunes software.
- × Some of the things the software might do include:
  - Download an MP3 music file and store it in the application's library.
  - Capture streaming music and store it in the application's library.
  - Manage the application's library (e.g., delete songs or organize them in playlists).
  - Burn a list of the songs in the library onto a CD.
  - Load a list of the songs in the library onto an iPod or MP3 player.
  - Convert a song from MP3 format to AAC format and vice versa.
- × A use case describes how a user interacts with the system by defining the steps

required to accomplish a specific goal (e.g., burning a list of songs onto a CD).

× A use case provides a big picture of the functionality of the system.


convert music file to new format
download music file & save to library
capture streaming music & save to library
organize the library
burn a list of songs to CD
load a list of songs to iPod
User


convert music file to new format
download music file & save to library
capture streaming music & save to library
organize the library
burn a list of songs to CD
load a list of songs to iPod
edit song list
<< include >>
<< include >>
<< include >>
User

SEQUENCE DIAGRAM

× A **sequence diagram** is used to show the dynamic communications between objects during execution of a task. It shows the temporal order in which messages are sent between the objects to accomplish that task.

× Sequence diagrams can be used to show the interactions in one use case or in one scenario of the software system.

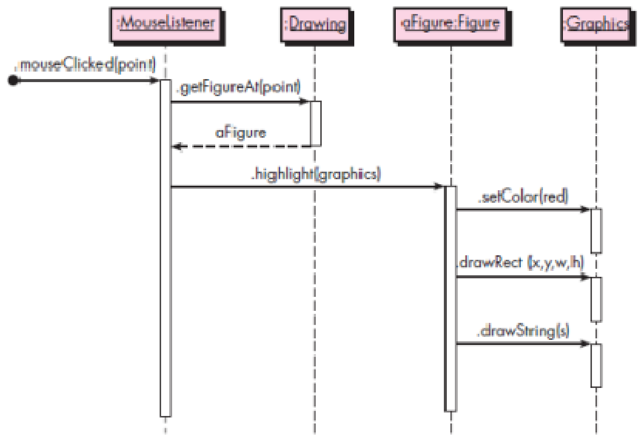


Figure 2.19
Example of sequence diagram of a drawing program

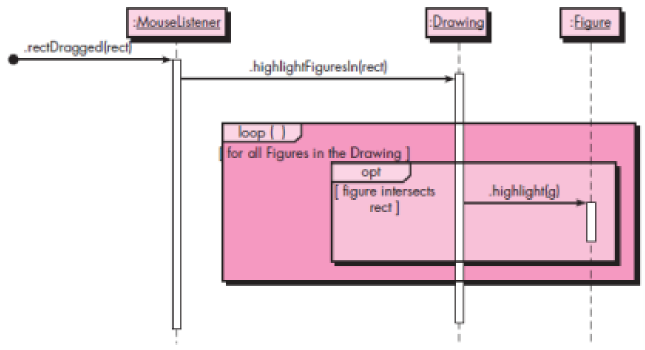

:MouseListener
:Drawing
:Figure
.rectDragged(rect)
.highlightFiguresIn(rect)
loop ( )
[for all Figures in the Drawing]
opt
[figure intersects rect]
.highlight(g)

Figure 2.20
Example of sequence diagram of a drawing program with loops

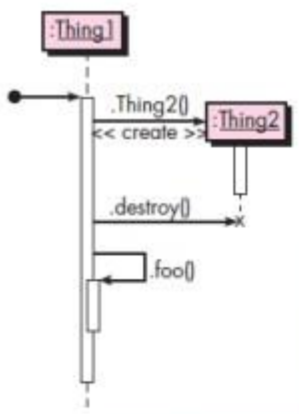


Figure 2.21
Special features included in a sequence diagram

SEQUENCE DIAGRAM: Another Example

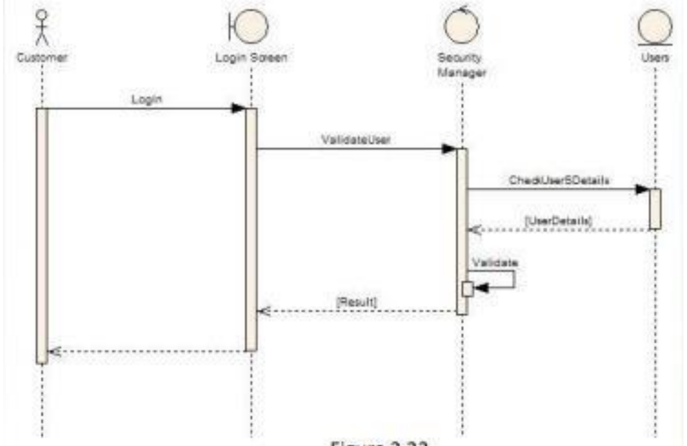


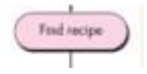
Figure 2.22
Another example of sequence diagram

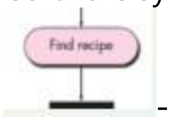
ACTIVITY DIAGRAM

× A **UML activity diagram** depicts the dynamic behavior of a system or part of a system through the flow of control between actions that the system performs.

× It is like a flowchart except that an activity diagram can show concurrent flows.

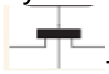
× Components:

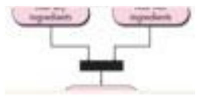
›  - **action node** (rounded rectangle) – tasked performed by the software system

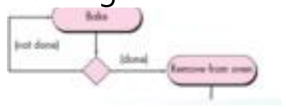
›  - **flow of control** (arrows) – arrow between two action nodes indicate that the first node will be performed following the second node.

› - **initial node** (solid black dot) – starting point

› - **final node** (a black dotsurrounded by a black circle) – end of activity

›  - **separation of activities** (fork) – consists of one arrow pointing to itand two or more arrows pointing out.

›  - **synchronization of concurrent flow of control** (join) – a horizontal black bar with two or more incoming arrows and one outgoing arrow

›  - **decision node**

(diamond with incoming arrow and twoor more outgoing arrows) – corresponds to a branch in the flow control based on a condition. Each outgoing arrow is labeled with a **guard** (a condition inside square brackets)

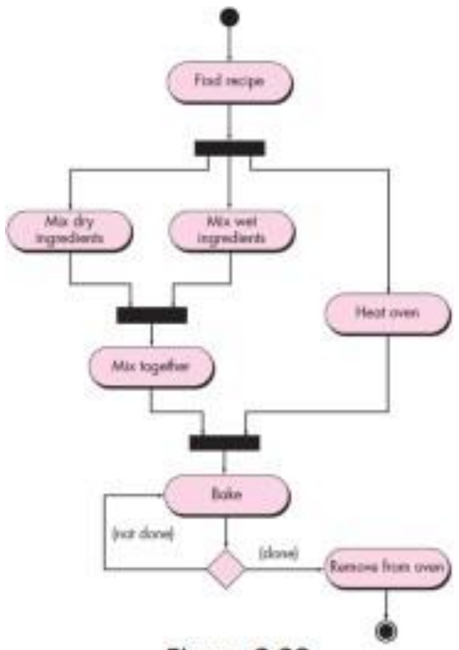
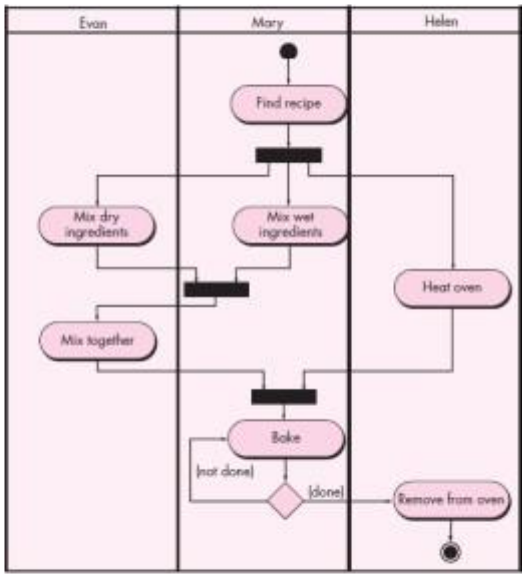


Figure 2.23

× Often, the exact division of labor does not matter. But if you do want to indicate how the actions are divided among the participants, you can decorate the activity diagram with **swimlanes**.

× Swimlanes, as the name implies, are formed by dividing the diagram into

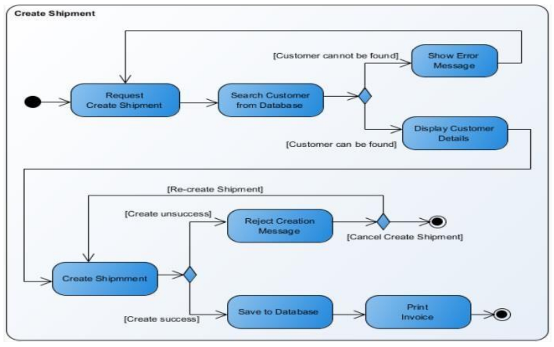
strips or “lanes,” each of which corresponds to one of the participants.



× An Activity Diagram resembles a horizontal flowchart that shows the actions and events as they occur. Activity diagrams show the order in which the actions take place and identify the outcomes.

× **Diagram Purpose**

› Activity Diagram is typically used for modeling the logic captured in aspecific use case in a use case diagram.Activity diagram can also be used to model a specific Actor's workflowwithin the entire system. Activity diagram can also be used independentof use cases for other purposes such asto model business process of a system,to model detailed logic of business rules etc. Activity diagram shows all potential sequence flows in an activity.


Create Shipment
Request Create Shipment
Search Customer from Database
[Customer cannot be found]
Show Error Message
[Customer can be found]
Display Customer Details
[Re-create Shipment]
[Create unsuccess]
Reject Creation Message
[Cancel Create Shipment]
Create Shipmment
[Create success]
Save to Database
Print Invoice