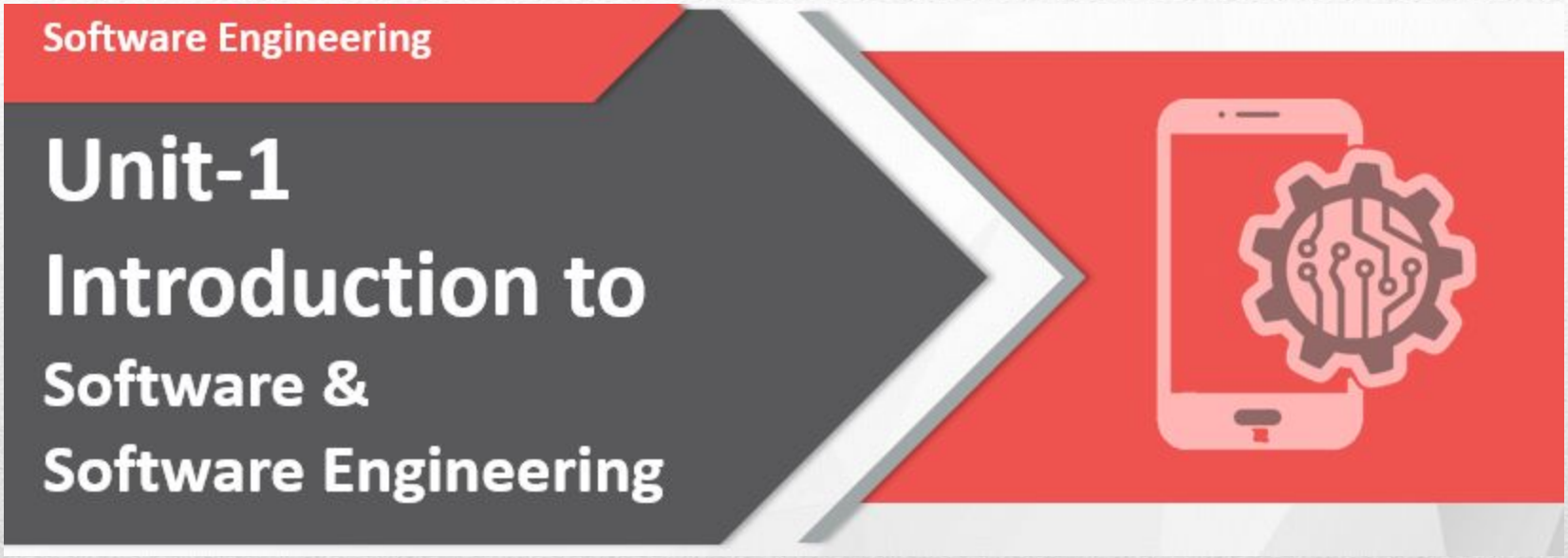
CSCA511: Software Engineering



Software Engineering: A Practitioner's Approach, 8/e
by Roger S. Pressman

Software Engineering 10/e
By Ian Sommerville

Software Engineering

Course Objectives, Outcomes & Syllabus

- Objectives:
  - To Understand the Significance of Software Development Process
  - To Introduce Different Software Life Cycle Models
  - To Design & Develop Robust Software Products
- Outcomes:
  - Ability to Understand various Phases of Software Development
  - Ability to Acquire Software Project Management Skills.
- Unit 1: Introduction to Software Engineering

Software Process Structure – Process Models & Activities – Agile Development – Requirements Engineering.

- Unit 2: Software Modelling

Design Concepts - Architectural Design - Component Level Design – User Interface Design – Web Application Design.

Software Engineering [Syllabus…]

- Unit 3: Quality Management

Review Techniques - Software Quality Assurance – Software Testing Strategies – Software Configuration Management – Product Metrics.

- Unit 4: Managing Software Projects

Project Management Concepts – Process and Project Metrics – Estimation for Software Projects – Project Scheduling - Risk Management.

- Unit 5: Reliability & Security

Reliability Engineering - Reliability & Availability – Reliability Testing. Security Requirements & Design.

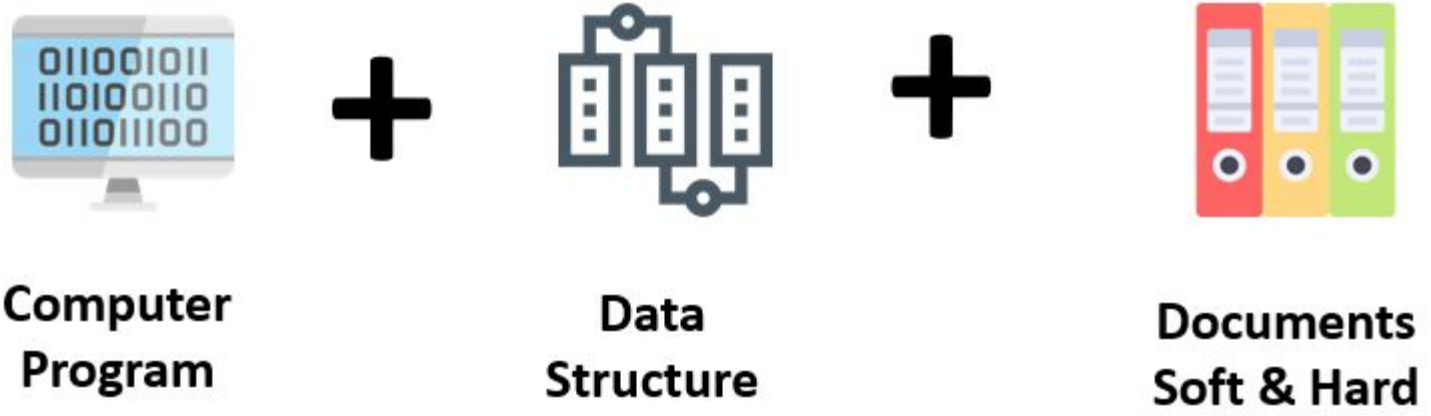
Software Engineering [Syllabus…]

Text Books:

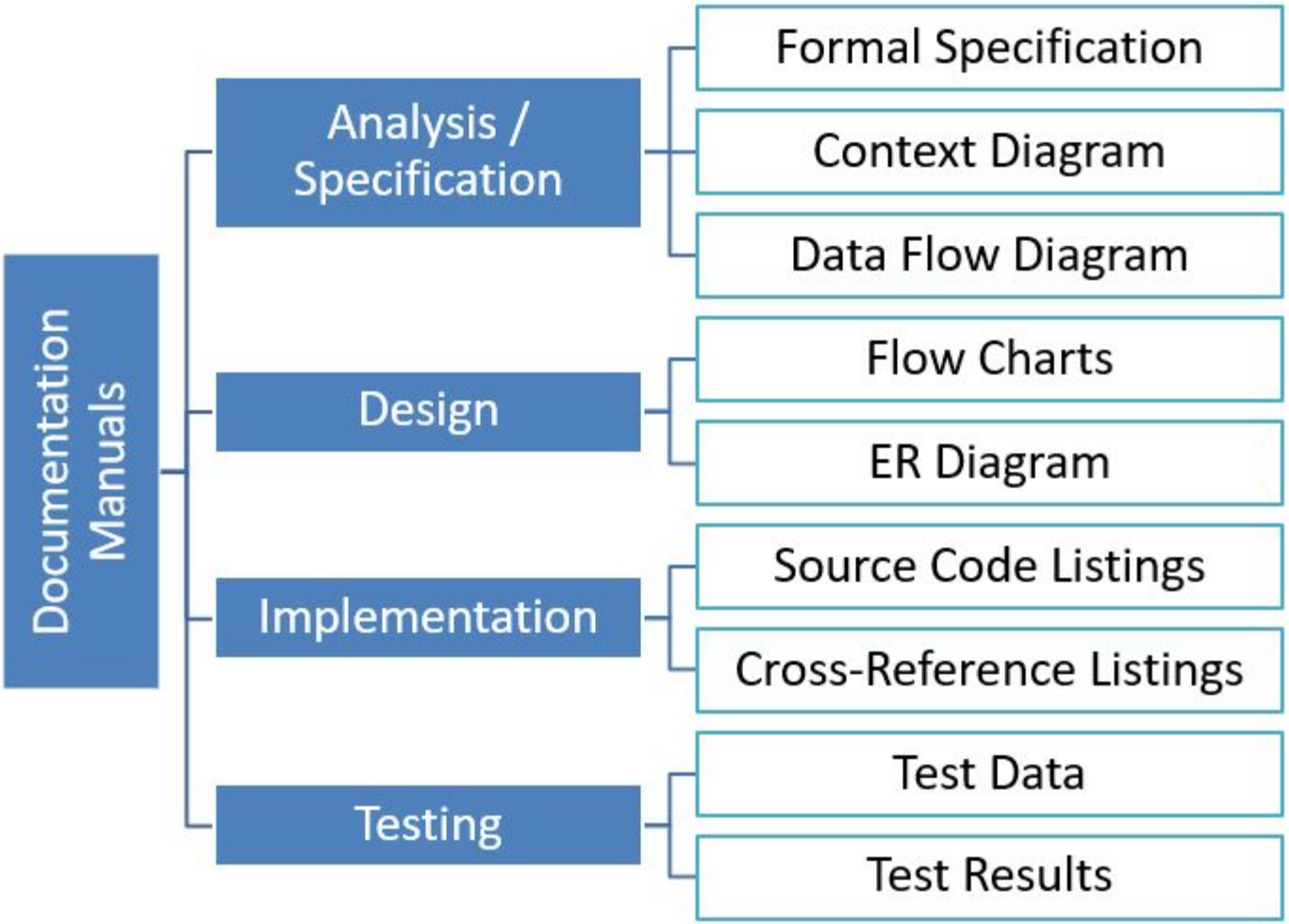
1. Software Engineering: A Practitioner's Approach, Roger S. Pressman, McGraw-Hill Education; 8th Ed., 2014.
2. Software Engineering, Ian Sommerville, Pearson Publishers, 10th Ed., 2015.

What is Software?

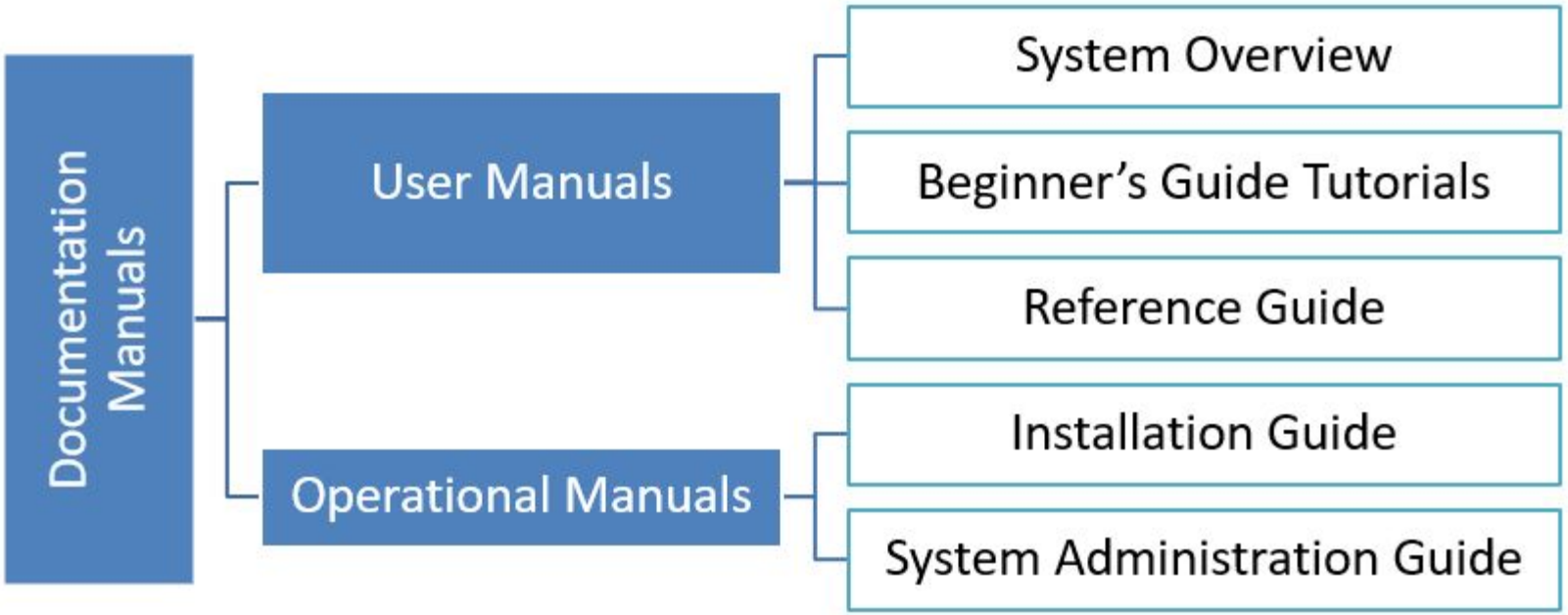
- Software is
  1) **Computer program** that when executed provide desired features, function & performance
  2) **Data Structure** that enable programs to easily manipulate information
  3) **Descriptive information** in both hard and soft copy that describes the operation and use of programs



List of Documentation Manuals



List of Documentation Manuals



Software Engineering

	Design	Build	Product
Engineering			
Software Engineering			

"Software Engineering" Definition

The IEEE Definition:

Software Engineering: The Application of a Systematic, Disciplined, Quantifiable Approach to the Development, Operation, and Maintenance of Software; that is, the Application of Engineering to Software.

Software Engineering

Software engineering is the establishment and use of **sound engineering principles** in order to obtain **economically software** that is **reliable and works** efficiently in **real machines**.

Software Engineering is the science and art of building (designing and writing programs) a software systems that are:

1) on time
2) on budget
3) with acceptable performance
4) with correct operation

FAQ about Software Engineering

Question	Answer
What is Software?	Computer programs, data structures and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good Software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software Engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What is the difference between Software Engineering and Computer Science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the Difference between Software Engineering and System Engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

Essential Attributes of Good Software

Product Characteristic	Description
Maintainability	Software should be written in such a way so that it "can evolve to meet the changing needs of customers". This is a critical attribute because Software Change is an inevitable Requirement of a changing business environment.
Dependability & Security	Software Dependability includes a range of characteristics including Reliability, Security and Safety. "*Dependable Software should not cause Physical or Economic Damage*" in the event of System Failure. "Malicious Users" should not be able to Access or Damage the System".
Efficiency	Software should not make wasteful use of System Resources such as "Memory and Processor" cycles. Efficiency therefore includes Responsiveness, Processing Time, Memory Utilisation, etc.
Acceptability	Software must be Acceptable to the Type of Users for which it is Designed. This means that it must be "*Understandable, Usable and Compatible"* with other Systems that they use.

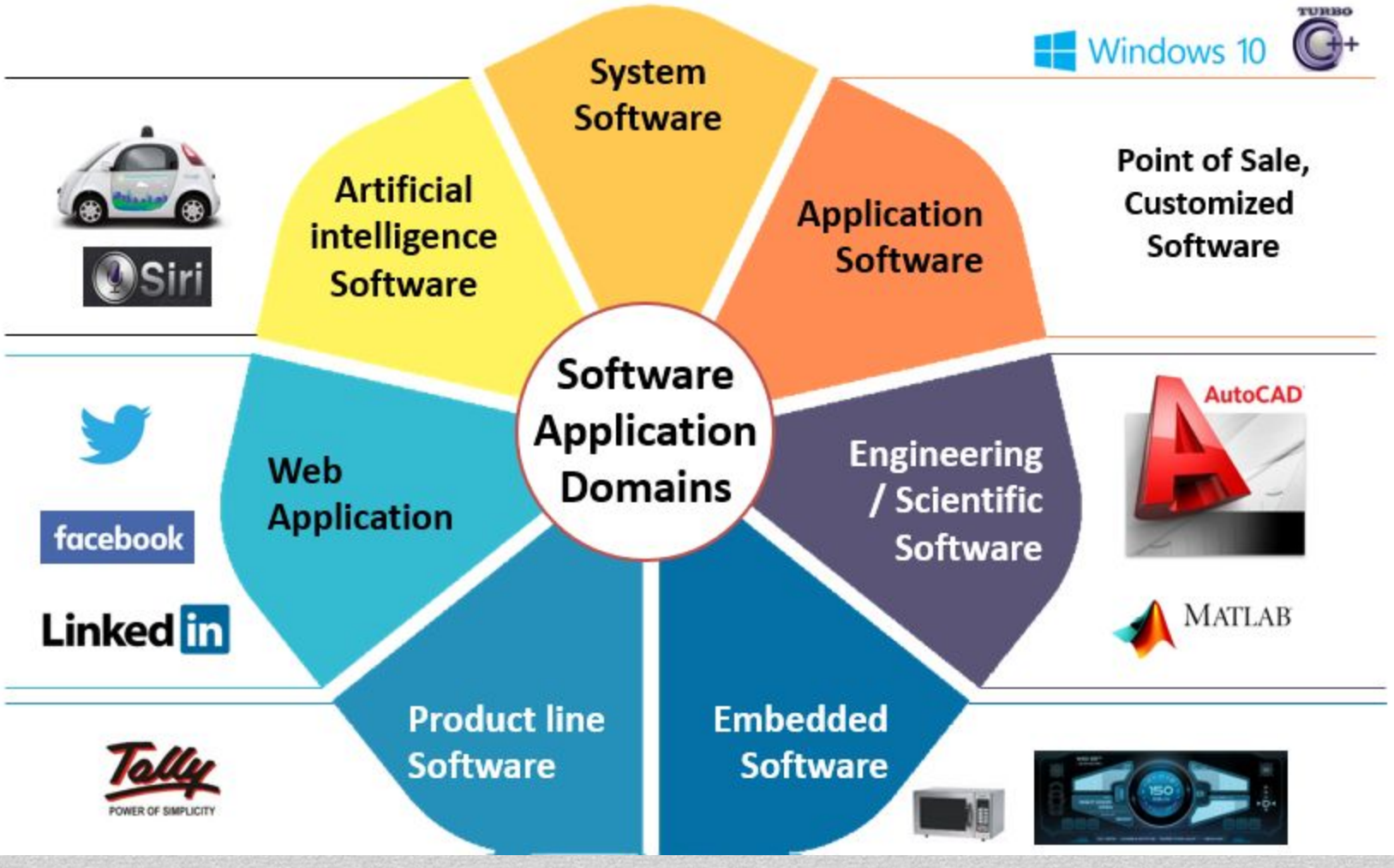
Software's Dual Role

- Software is a Product
  - *Transforms* information - Produces, Manages, Acquires, Modifies, Displays, or Transmits Information
  - Delivers Computing Potential of Hardware and Networks
- Software is a Vehicle for Delivering a Product
  - Controls other Programs (Operating System)
  - Effects Communications (Networking Software)
  - Helps Build other Software (Software Tools & Environments)

Software Products

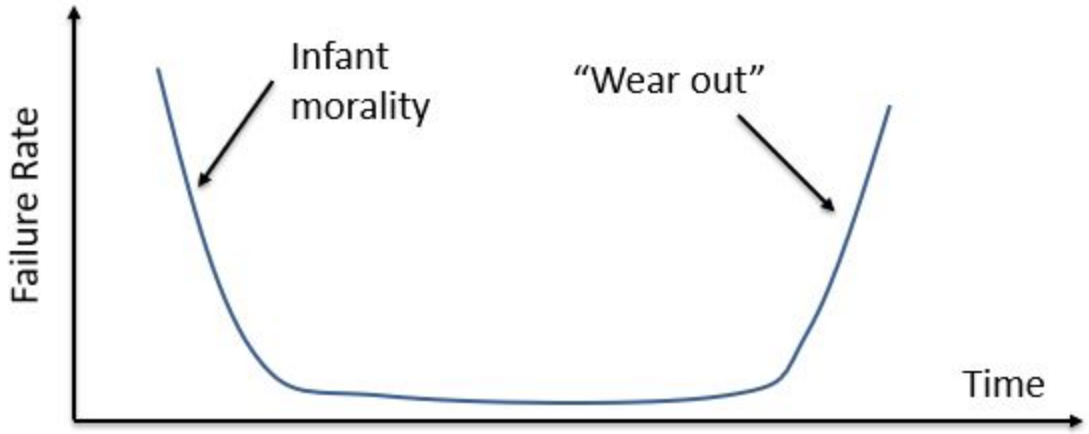
- **Generic Products**
  - Stand-alone systems that are marketed and sold to **any customer** who wishes to buy them.
  - Examples – PC software such as editing, graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- **Customized Products**
  - Software that is commissioned by **a specific customer** to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Software Application Domains



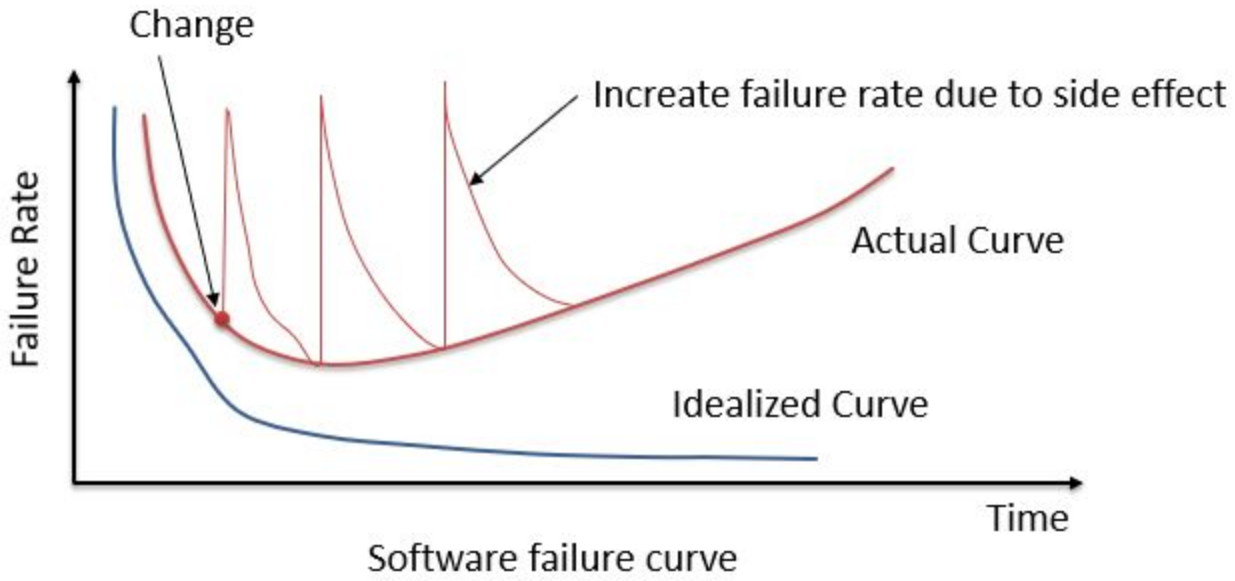
Characteristics of Software

- **Software is developed or engineered**
  - It is not manufactured like hardware
    - Manufacturing phase can introduce quality problem that are nonexistent (or easily corrected) for software
    - Both requires construction of "product" but approaches are different
- **Software doesn't "wear-out"**



Bathtub curve of hardware failure

Characteristics of Software cont.



Software failure curve

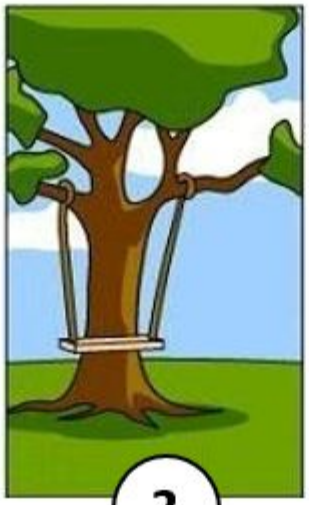
- Although the industry is **moving toward component based construction**, **most software** continues to be **custom built**

Why to Study Software Engineering?

Software Development Life Cycle without Software Engineering

1

How the Customer Explains Requirement

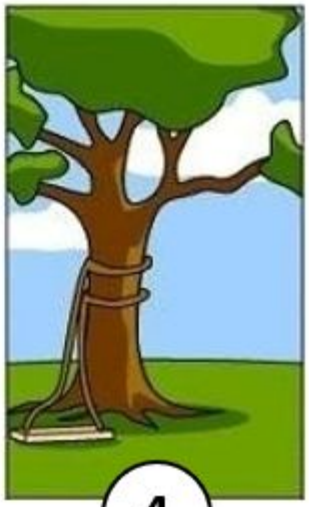


2

How the Project Leader understand it

3

How the System Analyst design it

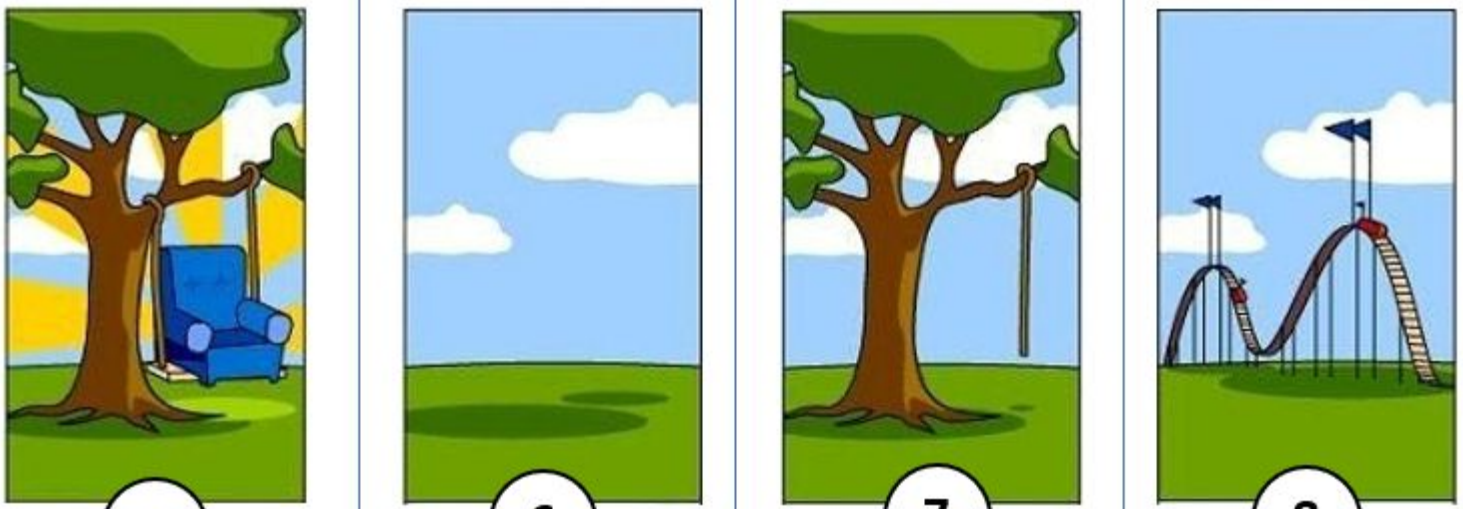


4

How the Programmer Works on it

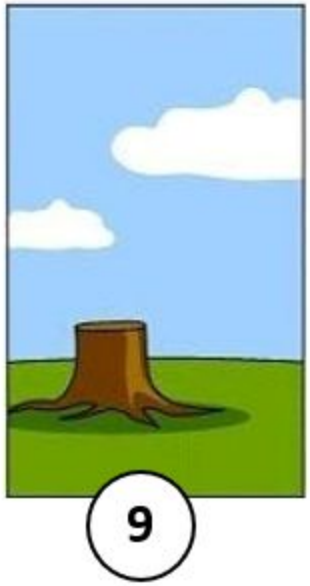
Why to Study Software Engineering?

Software Development Life Cycle without Software Engineering



Why to Study Software Engineering?

Software Development Life Cycle
without Software Engineering



9

How it was supported

10

What the customer really needed

Software development **Process** needs to be **engineered** to avoid the **communication gape** & to **meet the actual requirement** of customer within stipulated budget & time

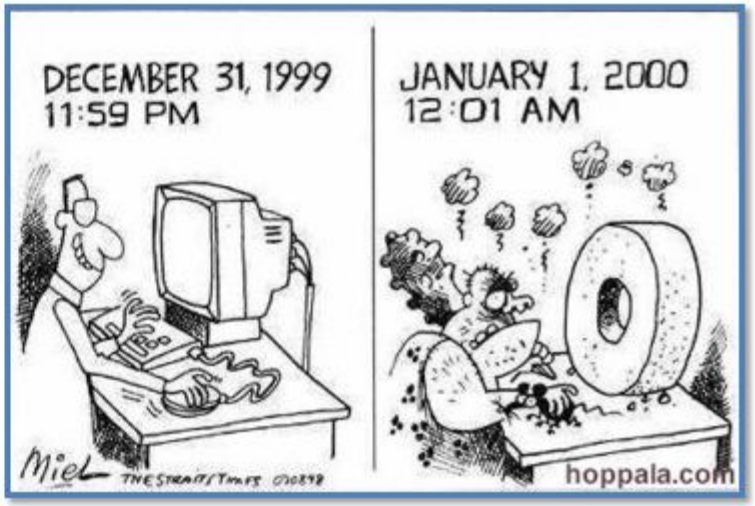
Ariane 5 - One bug, one crash

- It took the European Space Agency 10 years and **$7 billion** to produce Ariane 5, a giant rocket capable of hurling a pair of three-ton satellites into orbit with each launch and intended to give Europe overwhelming supremacy in the commercial space business.
- Issue was a small **computer program** trying to **stuff a 64-bit** number into a **16-bit space**.

One bug, one crash.
Of all the careless lines of code recorded
in the history of computer science

Y2K bug (millennium bug)

- The Y2K bug was a computer flaw, or bug, that may have caused problems when **dealing with dates** beyond December 31, 1999.
- The flaw, faced by computer programmers and users all over the world on January 1, 2000, is also known as the **"millennium bug**..
- Computer engineers used a ***two-digit code for the year***. Instead of a date reading 1970, it read 70.



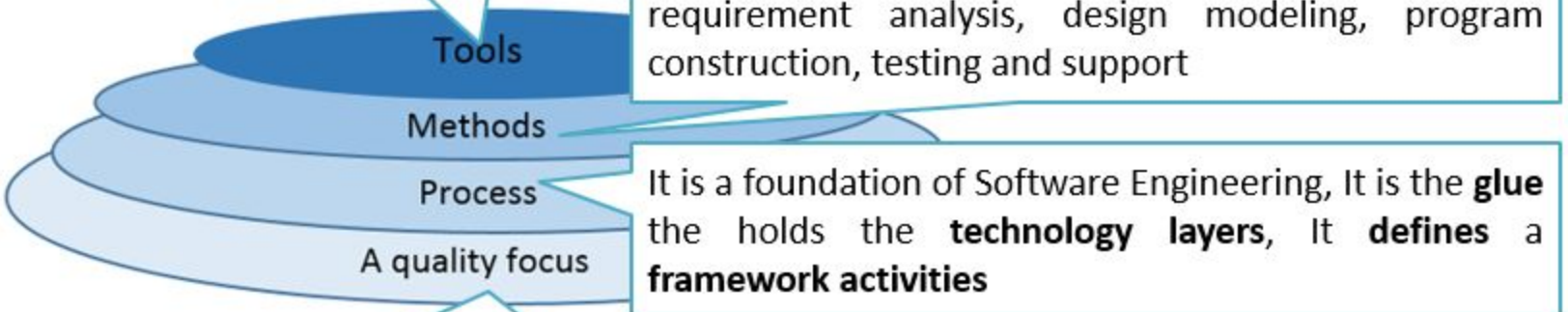
As the year 2000 approached, computer programmers realized that computers might not interpret
00 as **2000**, but as **1900**

SDLC without Software Engineering

Customer Requirement	Solution
• Have one trunk • Have four legs • Should carry load both passenger & cargo • Black in color • Should be herbivorous	• Have one trunk • Have four legs • Should carry load both passenger & cargo • Black in color • Should be herbivorous
	 Our value added Also gives milk

Software Engineering Layered Approach

Software Engineering Tools **allows automation of activities** which helps to perform systematic activities. A system for the support of software development, called **computer-aided software engineering** (CASE). **Examples:** Testing Tools, Bug/Issue Tracking Tools etc...



It provides **technical how-to's** for building software, it encompasses **many tasks** including communication, requirement analysis, design modeling, program construction, testing and support

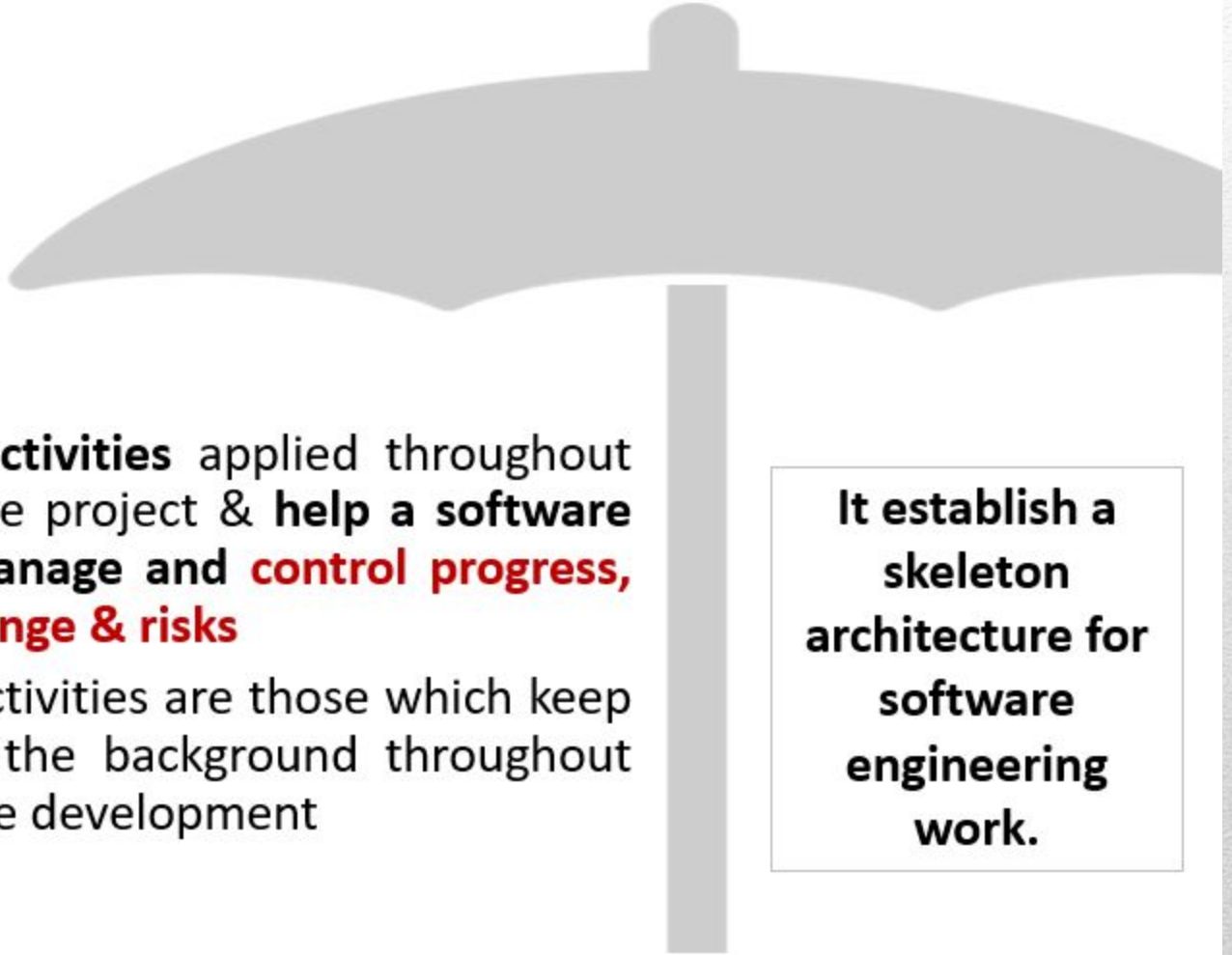
It is a foundation of Software Engineering, It is the **glue** the holds the **technology layers**, It **defines** a **framework activities**

Defines continuous **process improvement principles**

Process Framework Activities

Communication	Communication with Customers / stockholders to understand project requirements for defining software features	**Planning**	Software Project Plan which defines workflow that is to follow. It describes technical task, risks, resources, product to be produced & work schedule
Modeling	Creating models to understand requirements and shows design of software to achieve requirements	**Construction**	Code Generation (manual or automated) & Testing (to uncover errors in the code)
Deployment		Deliver Software to Customer Collect feedback from customer based on evaluation Software Support	

Umbrella Activities



- **Umbrella activities** applied throughout the software project & **help a software team to manage and control progress, quality, change & risks**
- Umbrella activities are those which keep running in the background throughout the software development

It establish a skeleton architecture for software engineering work.

Umbrella Activities Cont.

Software project Tracking & Control

Risk Management

Software quality assurance

Technical Reviews

Measurement

Reusability Management

Software Configuration Management

Work product preparation and production

Umbrella Activities Cont.

- **Software project tracking and control:** allows the software team to **assess progress against the project plan** and take any necessary action to maintain the schedule.
- **Risk management:** assesses (**evaluates**) **risks** that may affect the outcome of the project or the quality of the product.
- **Software quality assurance:** defines and conducts the activities required to **ensure software quality**.
- **Technical reviews:** **assesses** software engineering **work** products in an effort **to uncover** and remove **errors** before they are propagated to the next activity.
- **Measurement:** defines and collects process, project and product measures that assist the team in delivering software that meets stakeholders' needs.

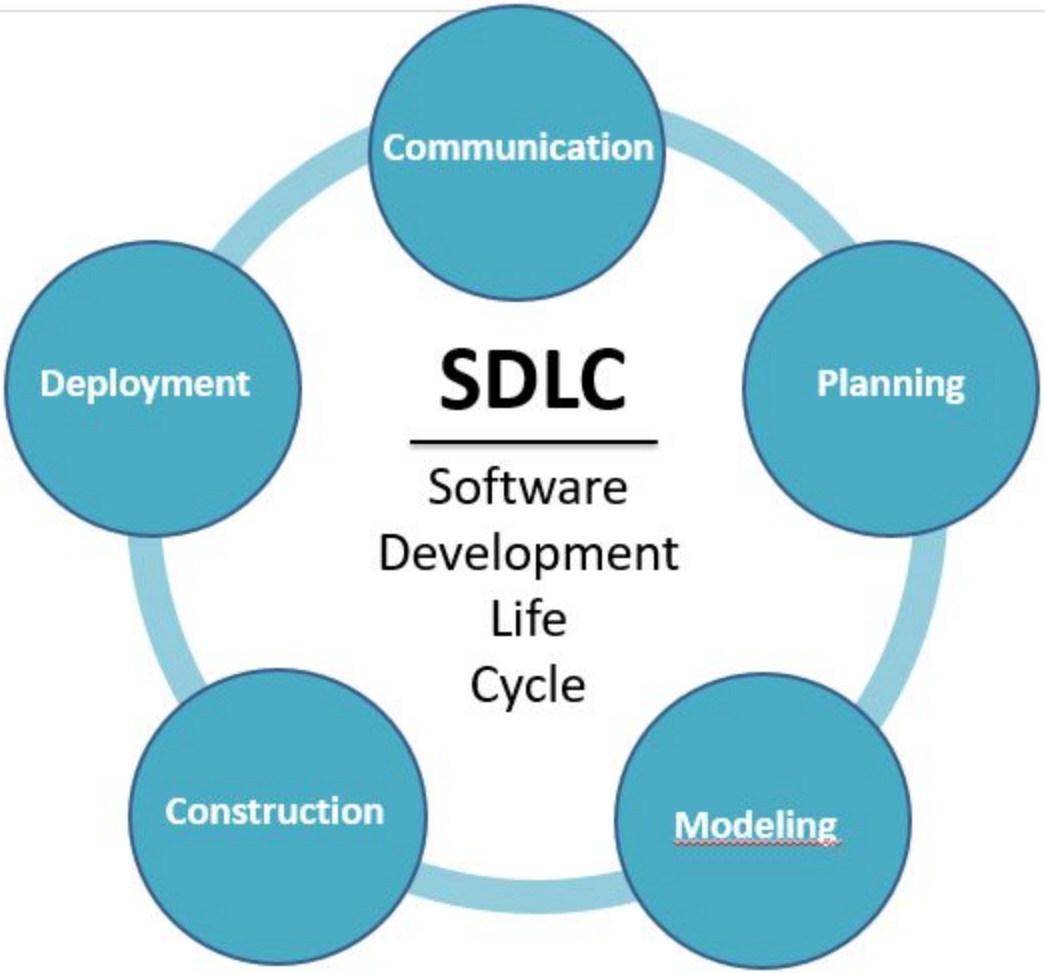
Umbrella Activities Cont.

- **Software configuration management:** it manages the effects of change throughout the software process.
- **Reusability management:** it defines criteria for work product reuse (including software components) and establishes **mechanisms to achieve reusable components**.
- **Work product preparation and production:** it encompasses (includes) the activities required to create work products such as **models, documents, logs, forms and lists**.

Software Process

- A **process** is a collection of **activities**, **actions** and **tasks** that are performed when some work product is to be created
- A process is not a **rigid prescription** for how to build the software
- Rather it is **adaptable approach** that enables the people doing the work to **pick** and **choose** the **appropriate set of work actions** and tasks
- The **purpose** of software process is
  - to **deliver** software in **timely** manner and
  - within sufficient **quality to satisfy** those
    - Who has given proposal for software development and
    - Those who will use software
  - A **process framework** establishes the foundation for complete software engineering process, it encompasses five activities

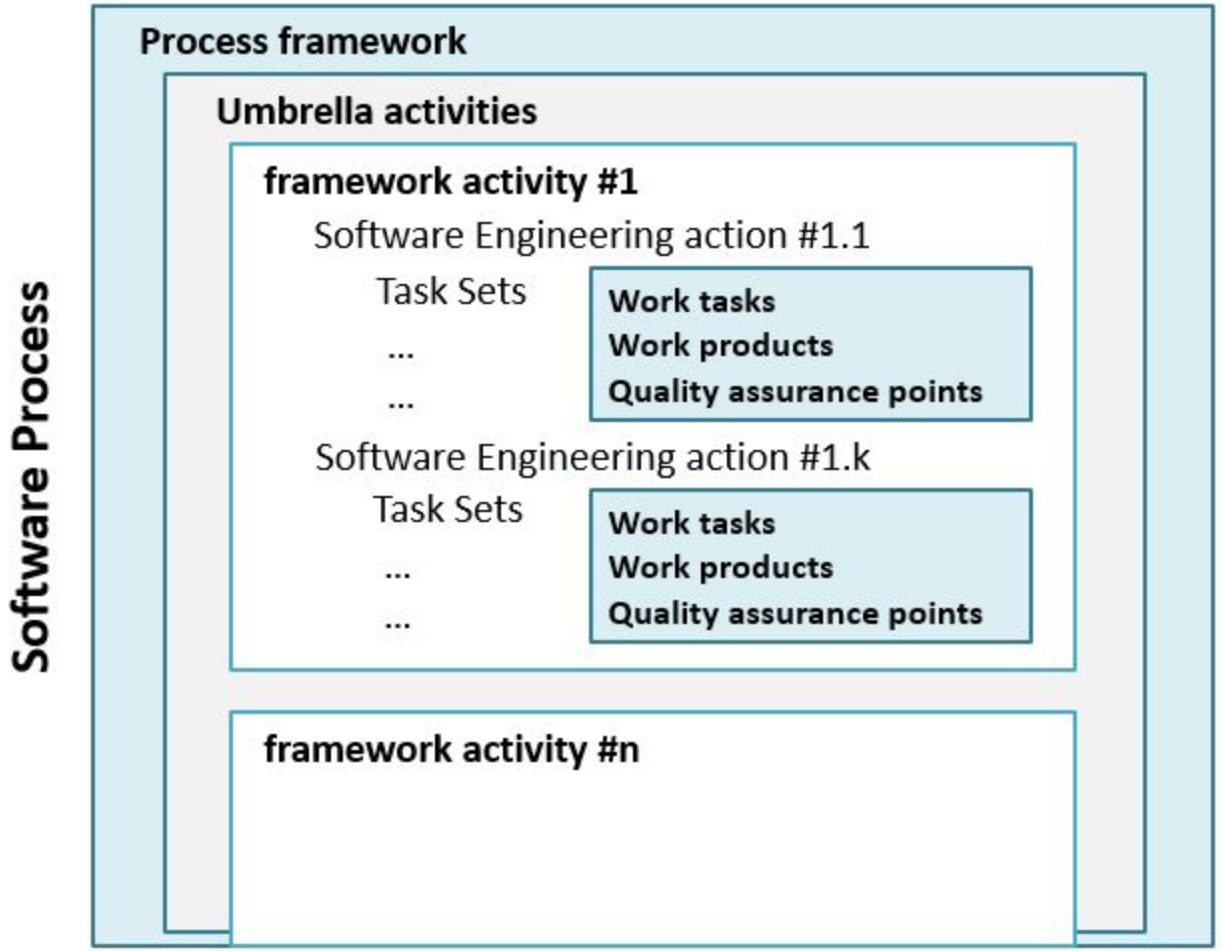
SDLC Phases


Communication
Planning
Modeling
Construction
Deployment
SDLC
Software
Development
Life
Cycle

Software Process Models

- The **process model** is the abstract representation of process.
- Also known as **Software development life cycle (SDLC)** or Application development life cycle Models
- Process models **prescribe** a distinct set of **activities, actions, tasks and milestones (deliverables)** required to engineer high quality software.
- Process **models are not perfect,** but **provide roadmap** for software engineering work.
- Software models provide stability, control and organization to a process that if not managed can easily get out of control.
- Software process models are adapted (adjusted) to meet the needs of software engineers and managers for a specific project.

Software Process Framework

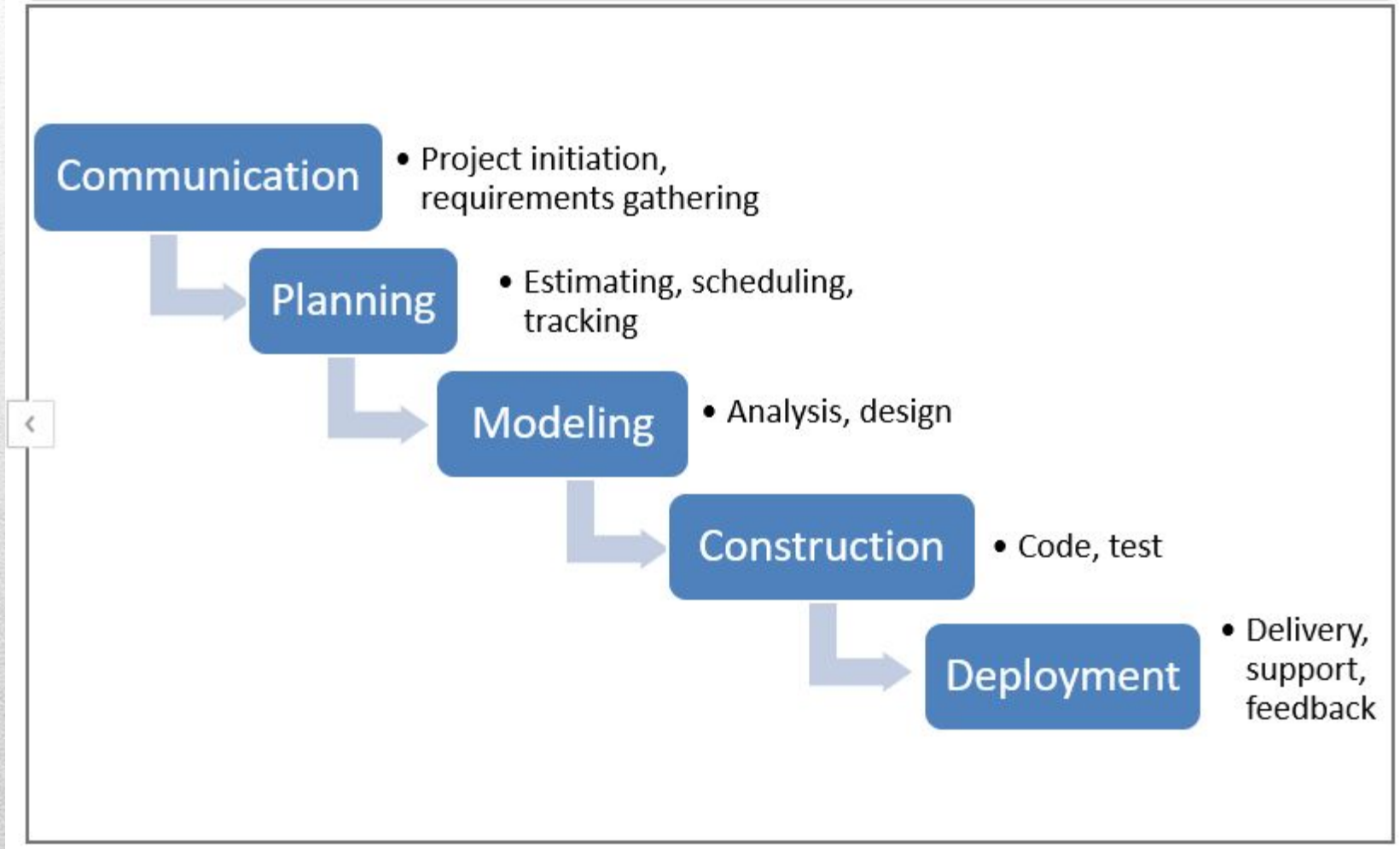


“Software Process Models”

Different Process Models

- Waterfall Model (Linear Sequential Model)
- Incremental Process Model
- Prototyping Model
- The Spiral Model
- Rapid Application Development Model
- Agile Model

The Waterfall Model



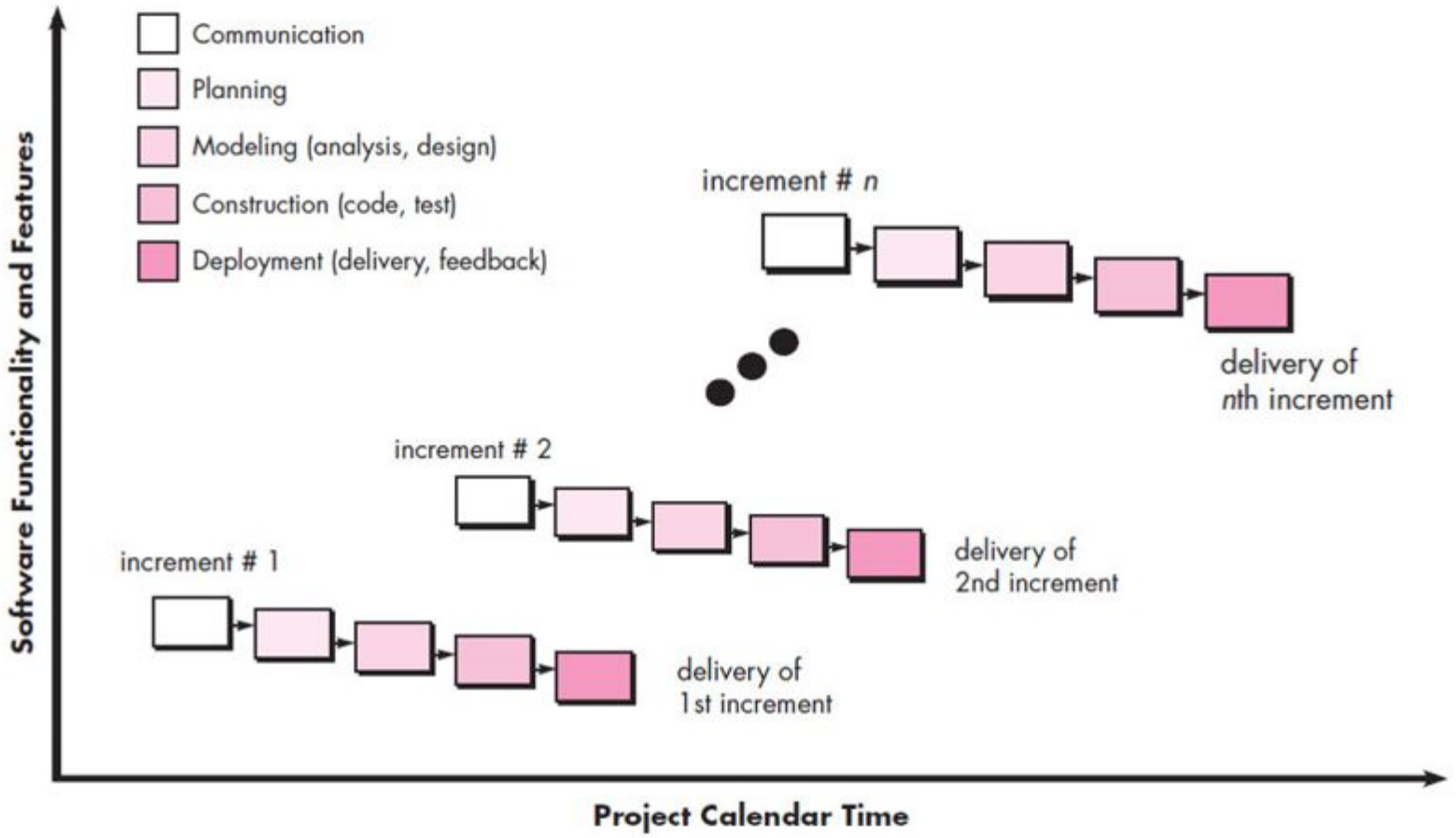
The Waterfall Model cont.

- When **requirements** for a problems are **well understood** then this model is used in which **work flow** from communication to deployment is **linear**
- This Model also called as the **Classic life cycle** or **linear sequential model**.
- **When to use ?**
  - Requirements are very well known, clear and fixed
  - Product definition is stable
  - Technology is understood
  - There are no ambiguous (unclear) requirements
  - Ample (sufficient) resources with required expertise are available freely
  - The project is short

The Waterfall Model cont.

- **Advantages**
  - **Simple to implement** and manage
- **Drawbacks**
  - **Unable to accommodate changes** at later stages, that is required in most of the cases.
  - **Working version** is **not available** during development. Which can lead the development with major mistakes.
  - **Deadlock can occur** due to delay in any step.
  - Not suitable for large projects.

Incremental Process Model



Incremental Process Model cont.

- The incremental model **combines** elements of **linear** and **parallel** process flows.
- This model applies linear sequence in a iterative manner.
- Initially **core working product** is **delivered**.
- **Each** linear **sequence** produces deliverable **"increments"** of the software.
- For example, word-processing software developed using the incremental model
  - It might deliver basic file management, editing and document production functions in the first increment
  - more sophisticated editing in the second increment;
  - spelling and grammar checking in the third increment; and
  - advanced page layout capability in the fourth increment.

Incremental Process Model cont.

- **When to Use ?**
  - When the **requirements** of the **complete** system are clearly **defined** and understood but **staffing is unavailable** for a **complete implementation** by the business deadline.
- **Advantages**
  - Generates **working software quickly** and early during the software life cycle.
  - It is **easier to test** and debug during a smaller iteration.
  - **Customer** can **respond** to each built.
  - **Lowers initial** delivery **cost**.
  - **Easier** to **manage risk** because risky pieces are identified and handled during iteration.

Evolutionary Process Models

- When a set of **core product** or system requirements is **well understood** but the **details of product** or system extensions have **yet to be defined**.
- In this situation there is **a need of process model** which specially designed to accommodate **product** that **evolve with time**.
- **Evolutionary Process Models** are specially meant for that which produce an increasingly more complete version of the software with each iteration.
- Evolutionary Models are **iterative**.
- Evolutionary models are
  - **Prototyping Model**
  - **Spiral Model**
  - **Concurrent Development Model**

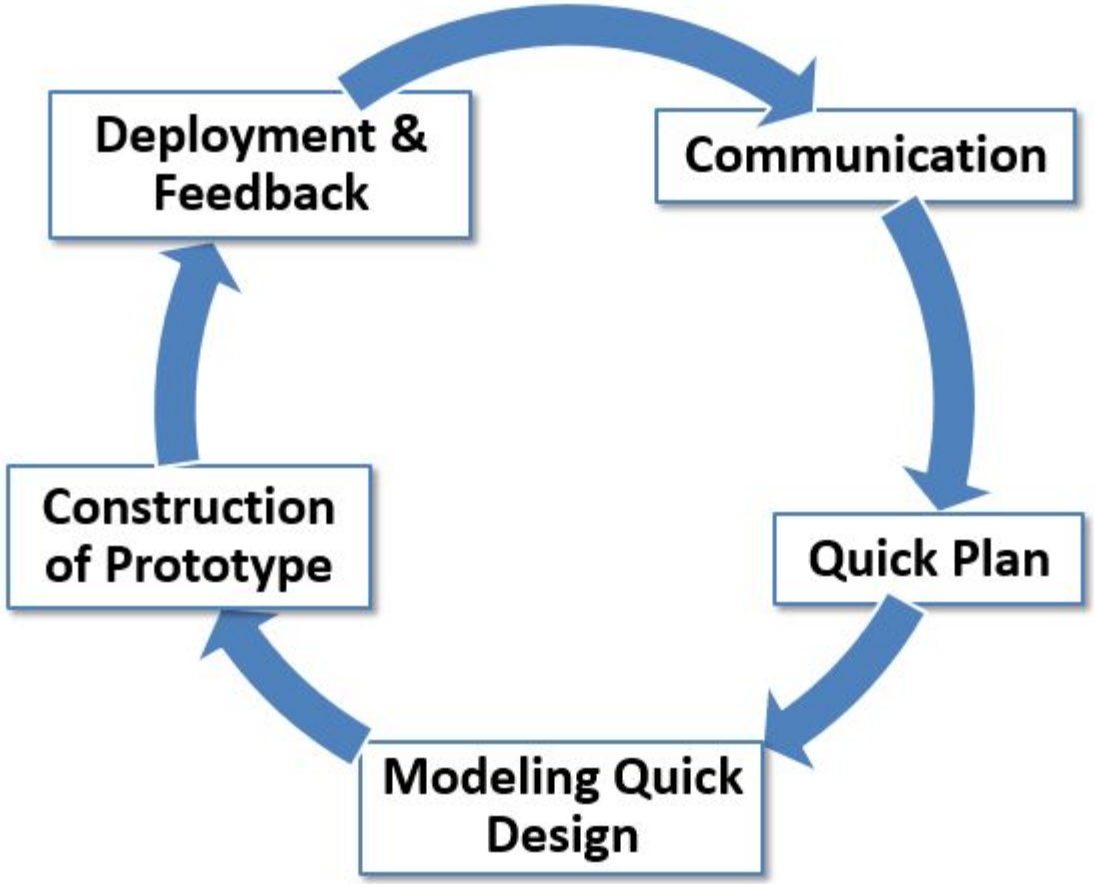
Prototyping model

- Prototyping model is appropriate when
  - **Customers have** general **objectives of software** but **do not have detailed requirements** for functions & features.
  - **Developers** are **not sure** about **efficiency of an algorithm** & **technical feasibilities**.
- It serves as a **mechanism** for **identifying software requirements**.
- Prototype can be serve as "**the first system**".
- Both stakeholders and software engineers like prototyping model
  - Users get feel for the actual system
  - Developers get to build something immediately

Prototyping model cont.

- **It works as follow**
  - **Communicate** with stockholders & **define objective** of Software
  - **Identify requirements** & design **quick plan**
  - **Model** a quick **design** (focuses on visible part of software)
  - **Construct Prototype** & deploy
  - Stakeholders **evaluate** this **prototype** and provides **feedback**
  - Iteration occurs and **prototype** is **tuned** based on **feedback**
- **Problem Areas**
  - **Customer demand** that "**a few fixes**" be applied to **make** the **prototype a working product**, due to that software quality suffers as a result
  - **Developer** often makes **implementation** in order to get a prototype working quickly; **without considering other factors** in mind like OS, Programming language, etc.

Prototyping model cont.



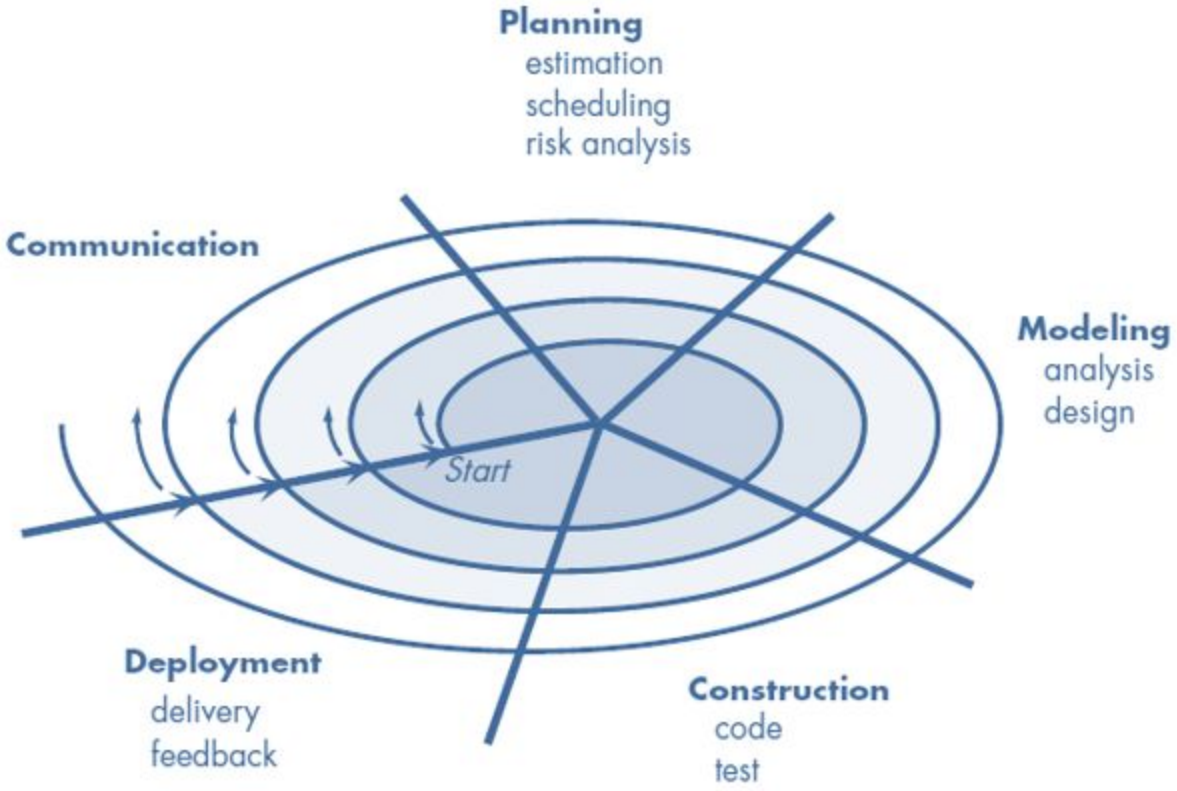
Prototyping model cont.

- **Advantages**
  - **Users** are actively **involved** in the **development**
  - Since in this methodology a working model of the system is provided, the **users get a better understanding** of the **system** being developed
  - **Errors** can be **detected** much **earlier**

The Spiral Model cont.

- The Spiral model is an **evolutionary process model** that couples **iterative nature of prototyping** with the **controlled and systematic aspects of waterfall** model
- It provides the **potential** for **rapid development**.
- Software is developed in a series of evolutionary releases.
- **Early iteration** release might be **prototype** but **later iterations** provides more **complete version of software**.
- It is divided into framework activities (C,P,M,C,D). Each activity represent one segment of the spiral
- **Each pass** through the **planning** region results in **adjustments** to
  - the **project plan**
  - **Cost** & **schedule** based on feedback

The Spiral Model



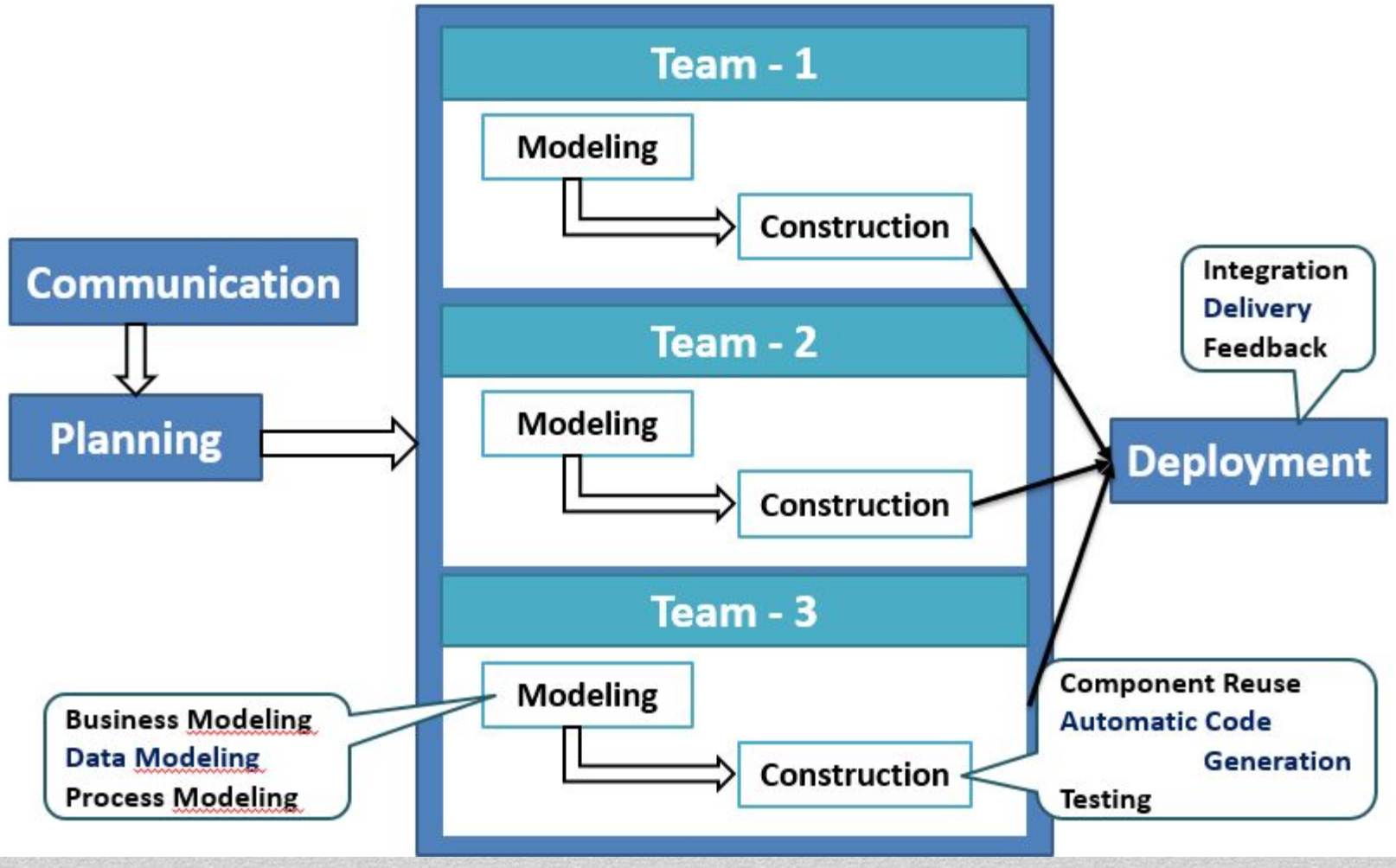
The Spiral Model cont.

- **When to use Spiral Model?**
  - For development of **large scale / high-risk projects**.
  - When costs and **risk evaluation is important**.
  - Users are **unsure** of their **needs**.
  - **Requirements** are **complex**.
  - New product line.
  - Significant (**considerable**) **changes** are expected.
- **Advantages**
  - High amount of risk analysis hence, **avoidance of Risk** is enhanced.
  - **Strong approval** and **documentation** control.
  - **Additional functionality** can be **added** at a later date.
  - **Software** is **produced early** in the Software Life Cycle.

The Spiral Model cont.

- **Disadvantages**
  - Can be **a costly model** to use.
  - Risk analysis **requires highly specific expertise**.
  - Project's success is highly dependent on the risk analysis phase.
  - Doesn't work well for smaller projects.

Rapid Application Development Model



RAD Model Cont.

- It is also know as **RAD** Model
- It is a type of **incremental model** in which; **components** or functions are **developed in parallel**.
- Rapid development is **achieved** by **component based construction**
- This can **quickly give** the customer **something to see** and use and to provide feedback.
- **Communication**
  - This phase is used to understand business problem.
- **Planning**
  - Multiple software teams work in parallel on different systems/modules.

RAD Model Cont.

- **Modeling**
  - **Business Modeling:** *Information flow* among the business.
    - Ex. What kind of information drives (moves)?
    - Who is going to generate information?
    - From where information comes and goes?
  - **Data Modeling:** Information refine into set of *data objects* that are *needed* to support business.
  - **Process Modeling:** *Data object* transforms **to** *information flow* necessary to implement business.
- **Construction**
  - It highlighting the *use of pre-existing software component*.
- **Deployment**
  - Deliver to customer basis on subsequent iteration.

RAD Model Cont.

- **When to Use ?**
  - There is a need to create a **system** that can be **modularized in 2-3 months** of time.
  - **High availability** of **designers** and **budget** for modeling along with the cost of automated code generating tools.
  - **Resources** with **high** business **knowledge** are available.
- **Advantages**
  - **Reduced** development **time**.
  - **Increases reusability** of components.
  - **Quick** initial **reviews** occur.
  - **Encourages** customer **feedback**.
  - Integration from very beginning **solves** a lot **of integration issues**.

RAD Model Cont.

- **Drawback**
  - For **large** but scalable **projects**, RAD **requires sufficient human resources**.
  - Projects **fail if developers** and **customers** are **not committed** in a much shortened time-frame.
  - **Problematic** if system **can not be modularized**.
  - **Not appropriate when technical risks are high** (heavy use of new technology).

Component based Development

- **Commercial off the shelf (COTS)** software **components** are offered **as product**.
- **COTS** provides **set of functionality** with **well defined interfaces** that enables component to be integrated into software.
- The component based development model **incorporates** many **characteristics** of the **spiral model**.
- It is **evolutionary** in **nature**.
- Component based development model **constructs** applications from **prepackaged** software **components**.
- **Modeling** and **construction** activities begin with the **identification of components**.

Component based Development cont.

- **Component based development incorporates the following steps**
  1. Available **component-based products** are **researched** & **evaluated** for software development .
  2. Component **integration issues** are **considered**
  3. A **software architecture** is **designed** to accommodate the components.
  4. Components are **integrated** into the **architecture**.
  5. **Testing** is conducted to insure proper functionality.
- **Advantages**
  - It leads to **software reuse**.
  - It **reduces development** cycle **time**.
  - **Reduction** in project **cost**.

Unified Modeling (UML)

Use-Cases

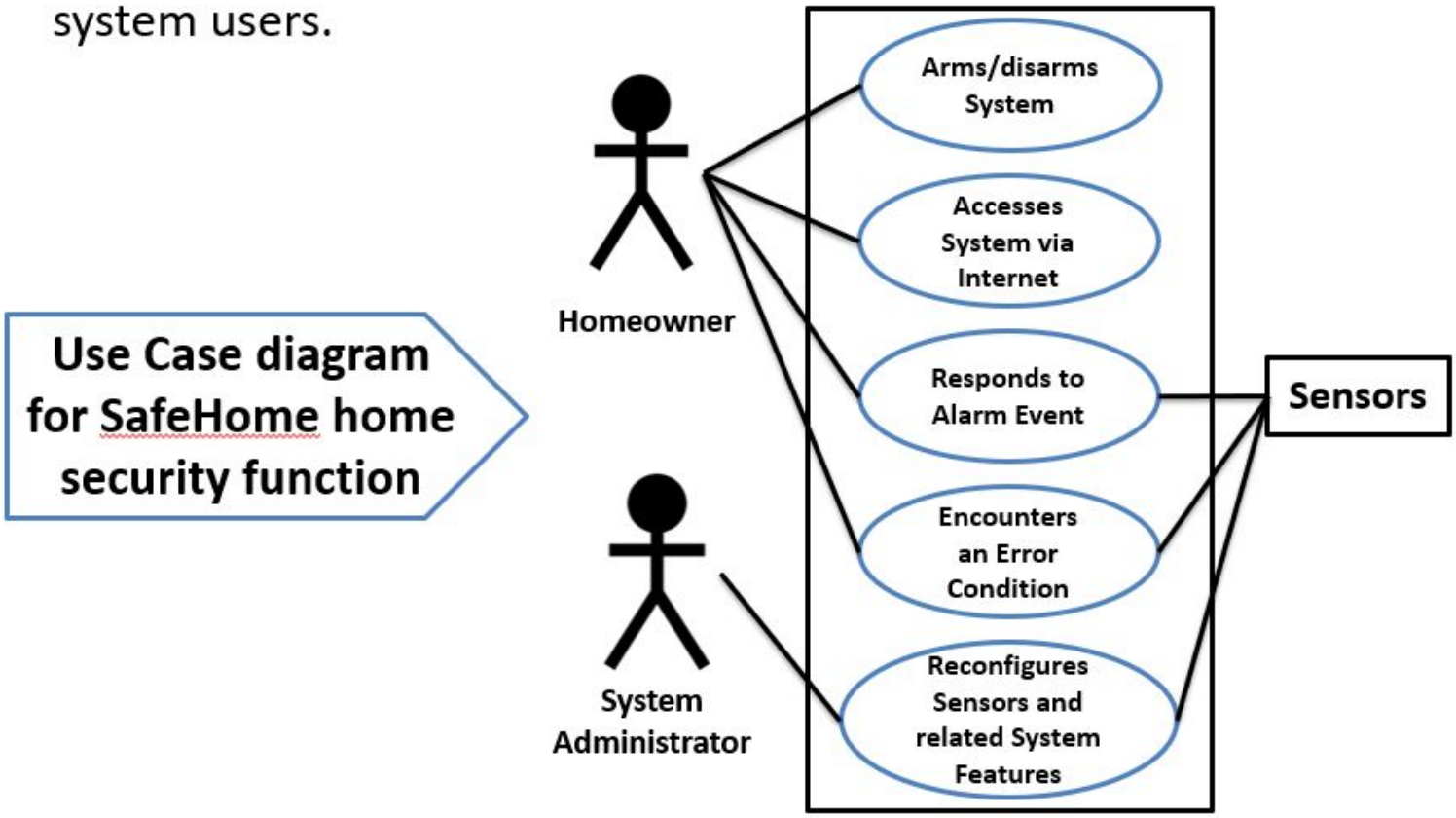
- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an "actor"
  - Actor: a person or device that interacts with the software
- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor (s)?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What extensions might be considered as the story is described?

Use-Cases

- What variations in the actor's interaction are possible?
- What system information will the actor acquire, produce, or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Use-Case Diagram

- It is referred as the diagram used to describe a set of actions (use cases) that some system should perform in collaboration with system users.

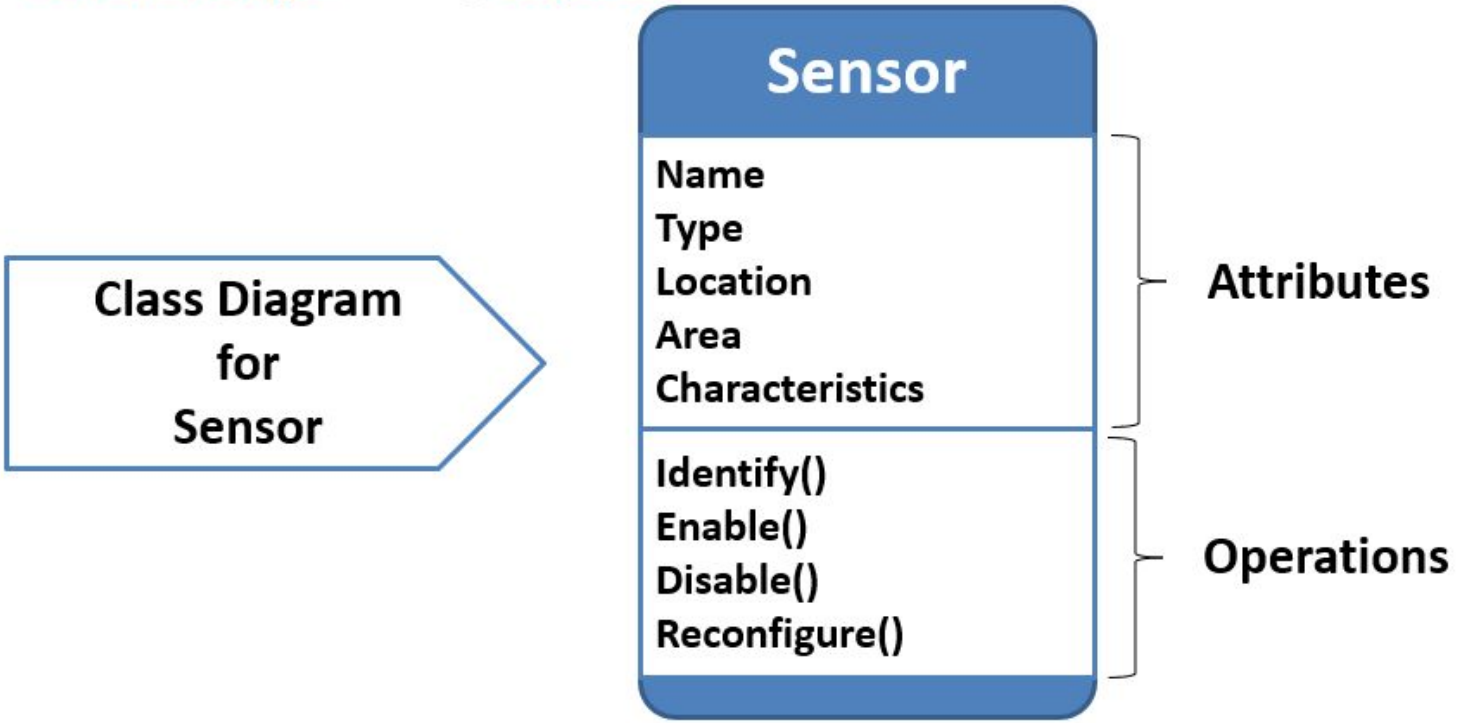


Use-Case

1	**Use Case Title**	**Login**
2	**Abbreviated Title**	Login
3	**Use Case Id**	1
4	**Actors**	Librarian , Members, Asst. Librarian
5	**Description:** To interact with the system, LMS will validate its registration with this system. It also defines the actions a user can perform in LMS.	
5.1	**Pre Conditions:** User must have proper client installed on user terminal	
5.2	**Task Sequence** 1. System show Login Screen 2. User Fill in required information. Enter user name and password 3. System acknowledge entry	
5.3	**Post Conditions:** System transfer control to user main screen to proceed further actions	
5.4	**Exception:** If no user found then system display Invalid user name password error message and transfer control to Task Sequence no.1	
6	**Modification history:** Date 08-01-2018	
7	**Author:** Pradyumansinh Jadeja Project ID LMS	

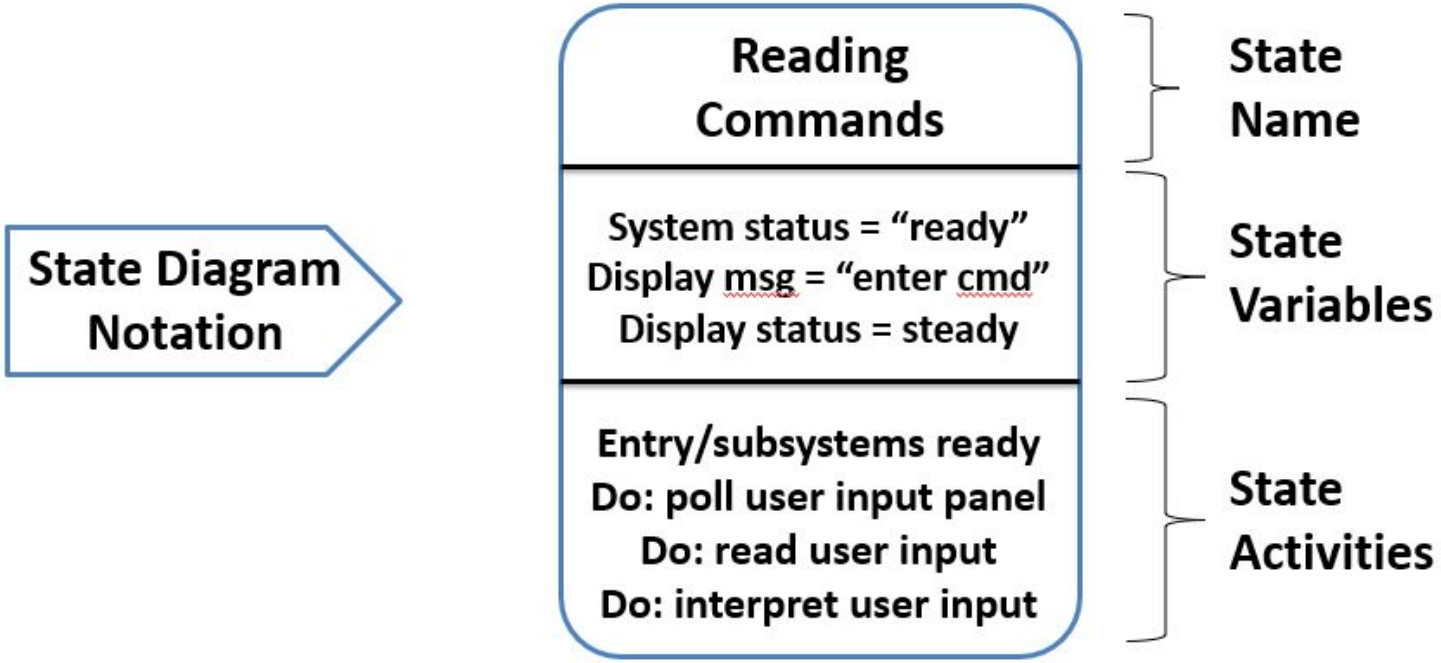
Class Diagram

- It **describes** the structure of a system by showing the **system's classes**, their **attributes**, **operations** (or methods), and the **relationships** among **objects**.



State Diagram

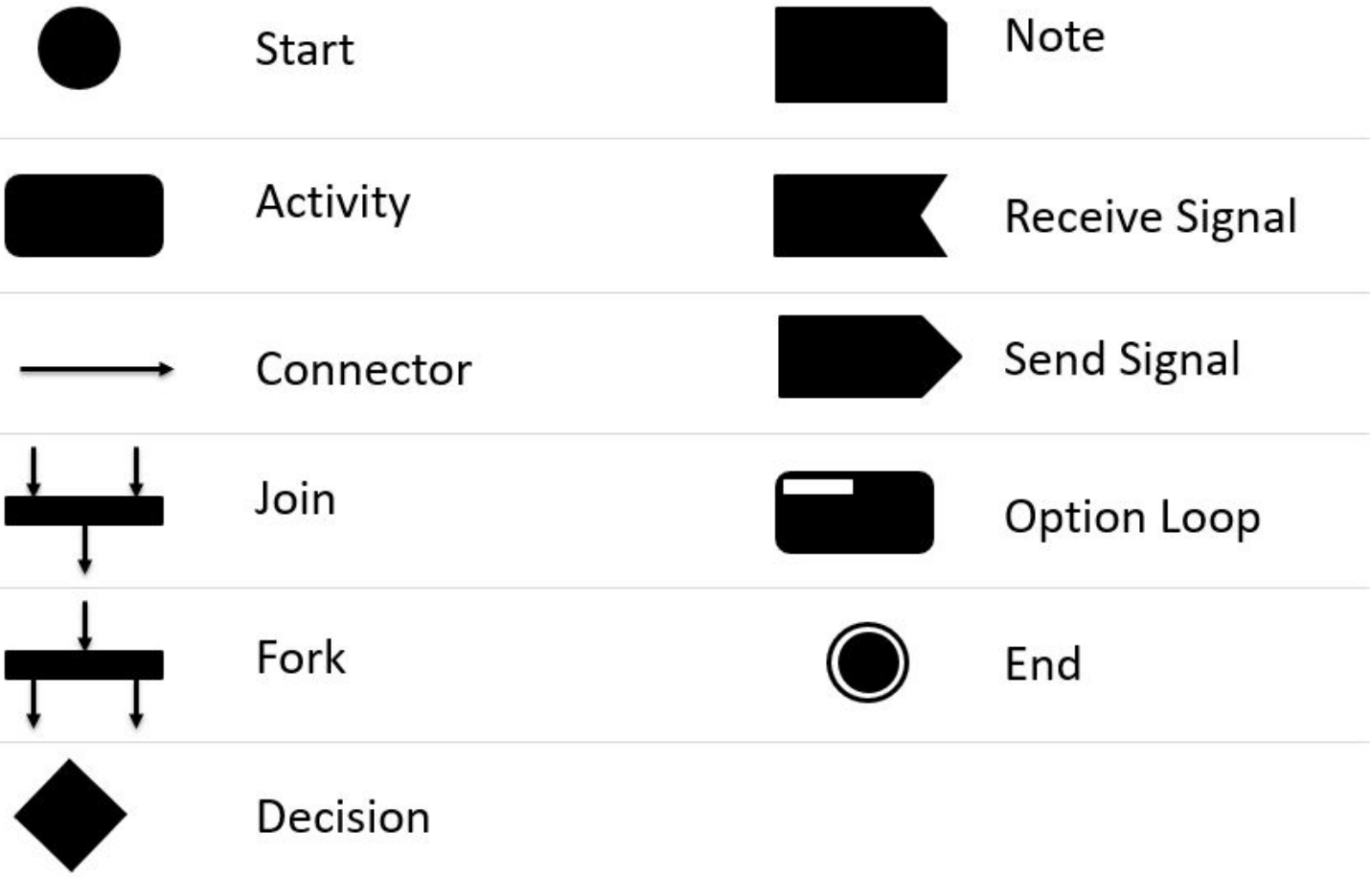
- It is used to **describe** the **behaviour** of **systems**.
- It requires that the system described is composed of a finite number of states.



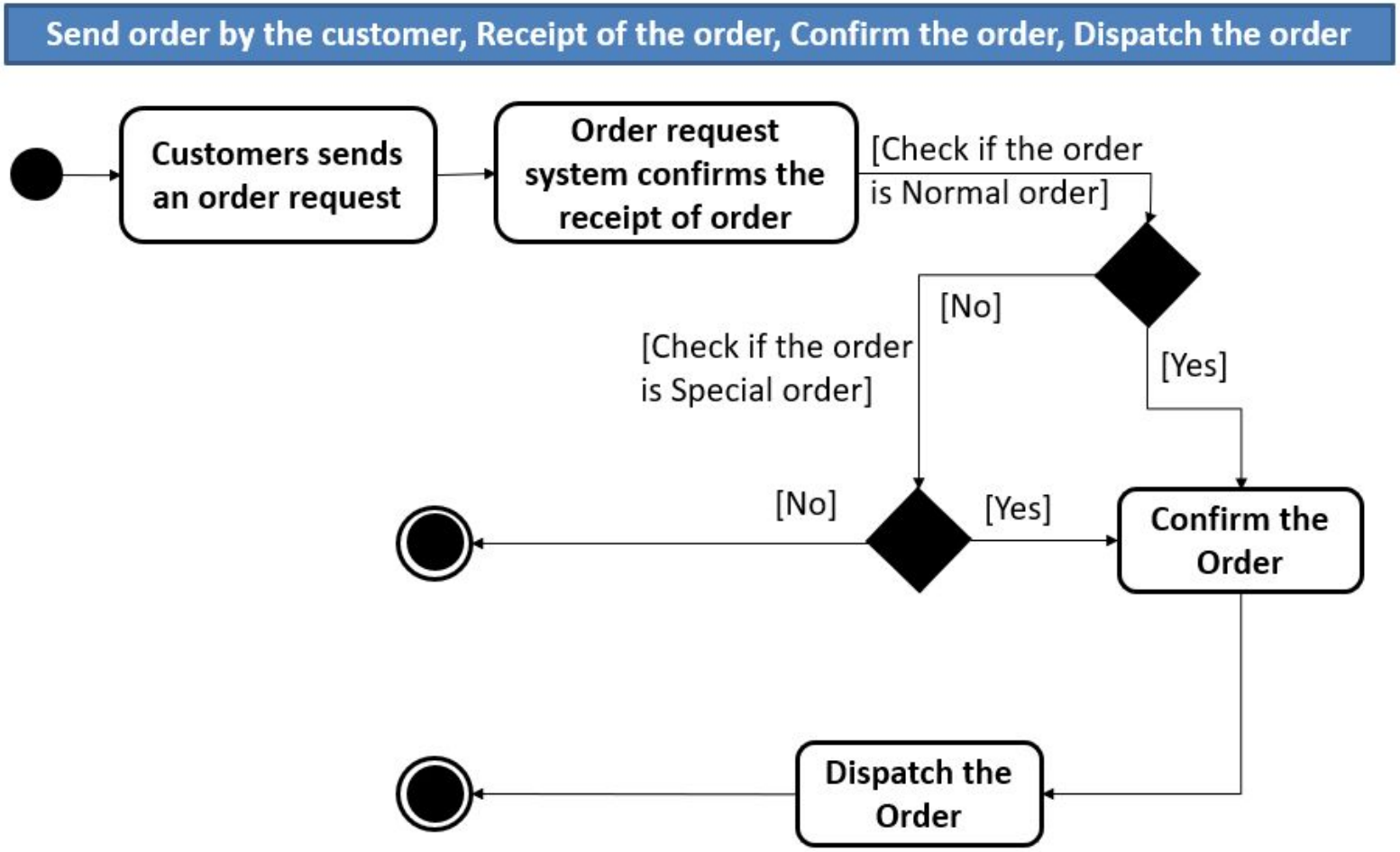
Activity & Swimlane Diagram

- **Activity diagram** is basically a **flowchart** to **represent** the **flow** from one **activity** to another **activity**
- The activity can be described as an operation of the system.
- A **swimlane diagram** is a type of activity diagram. Like activity diagram, it diagrams a process from start to finish, but it also **divides** these **steps** into **categories** to help **distinguish** which departments or employees are **responsible** for each set **of actions**
- A swim lane diagram is also useful in helping **clarify responsibilities** and help departments work together in a world where departments often don't understand what the other departments do

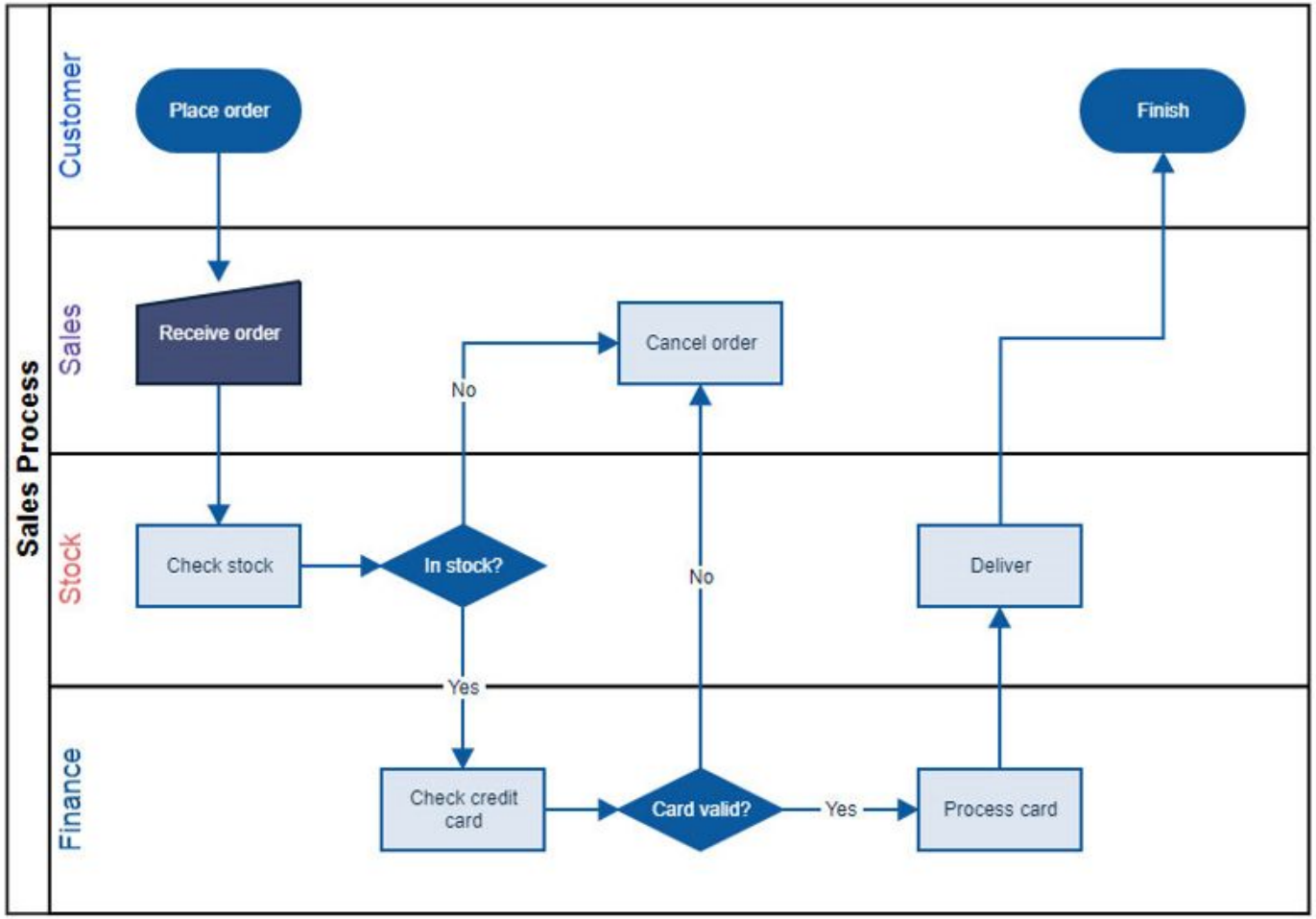
Activity Diagram Symbols



Activity diagram of order processing

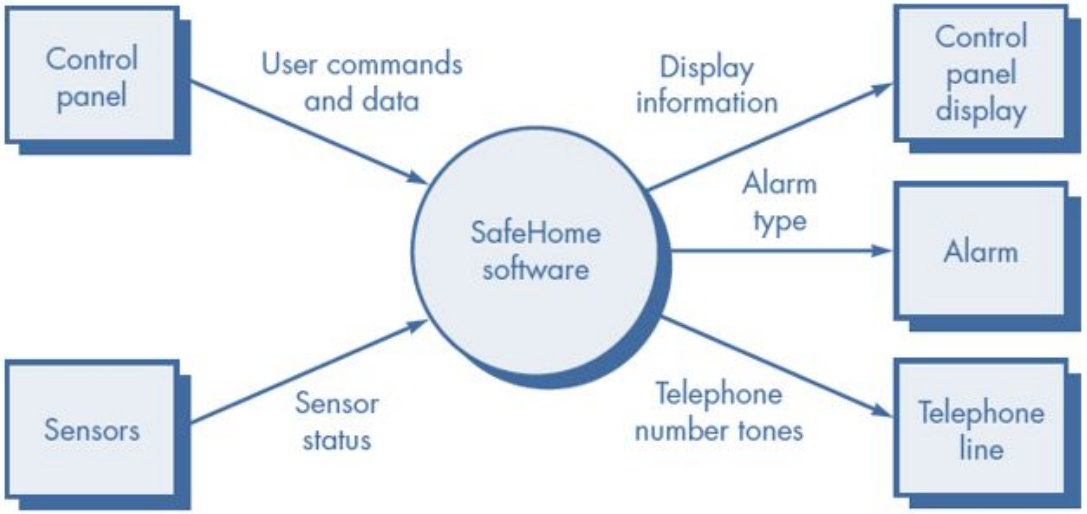


Swimlane diagram of order processing



Data Flow Diagram (DFD)

- It is a graphical representation of the "flow" of data through an information system, modelling its process aspects
- It is often **used** as a preliminary step to create an overview of the system, which can later be elaborated



Context-level DFD for the SafeHome security function

CSCA511: Software Engineering

Software Engineering: A Practitioner's Approach, 8/e
by Roger S. Pressman

Software Engineering 10/e
By Ian Sommerville

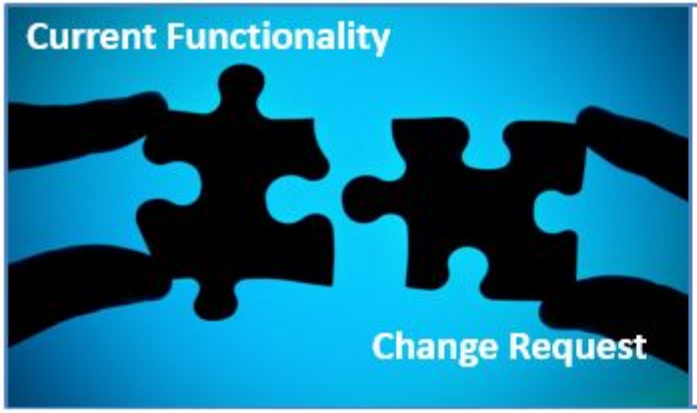
Outline

- Agility and Agile Process Model
- Extreme Programming
- Other Process Model of Agile Development and Tools
  - Adaptive Software Development (ASD)
  - Dynamic Systems Development Method (DSDM)
  - Scrum
  - Feature Driven Development (FDD)
  - Crystal
  - Agile Modelling (AM)

Agility

- Agility is **ability to move quickly and easily**.
- It is a property consisting of **quickness**, **lightness**, & **ease of movement**;
- The ability to **create** and **respond to change** in order to profit in a turbulent global business environment
- The ability to **quickly reprioritize use of resources** when requirements, technology, and knowledge shift
- A very **fast response to sudden** market **changes** and emerging threats by intensive customer interaction
- Use of **evolutionary, incremental,** and **iterative delivery** to converge on an optimal customer solution
- Maximizing **BUSINESS VALUE** with **right sized**, **just- enough**, and **just-in-time processes** and **documentation**

What is Agility?



Effective response to change

Organizing a team so that it is in control to perform the work

Effective communication among all stakeholders

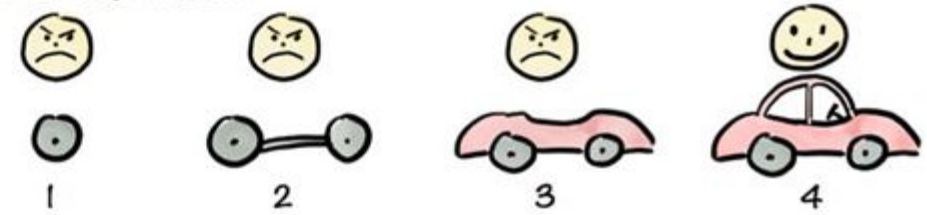
What is Agility? Cont.

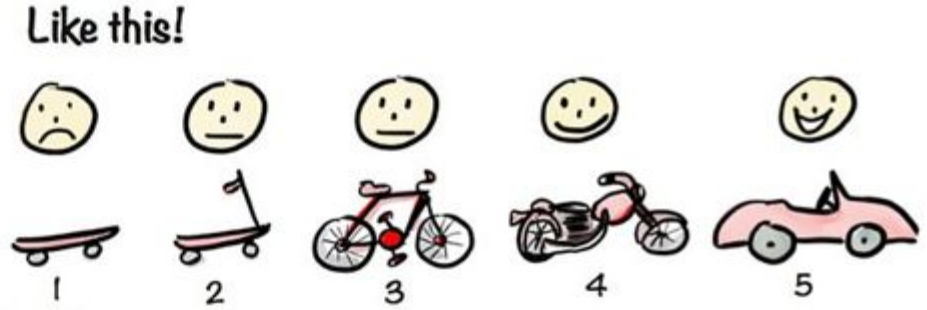
Software Development Team

Drawing the customer onto the team

Eliminate the "us and them" attitude

Rapid and Incremental delivery of software





Agility Principles

- **Highest priority** is to **satisfy** the **customer** through early & **continuous delivery** if **software**
- **Welcome changing** requirements
- **Deliver** working **software frequently**
- **Business people** and **developers** must **work together**
- **Build** projects **around motivated** individuals
- Emphasize **face-to-face conversation**
- **Working software** is the **measure** of **progress**
- Continuous **attention** to **technical excellence** and **good design**
- **Simplicity** – the art of maximizing the amount of work done
- The best designs emerge from **self-organizing teams**
- The **team tunes** and **adjusts** its **behaviour** to become more effective

Agile Process

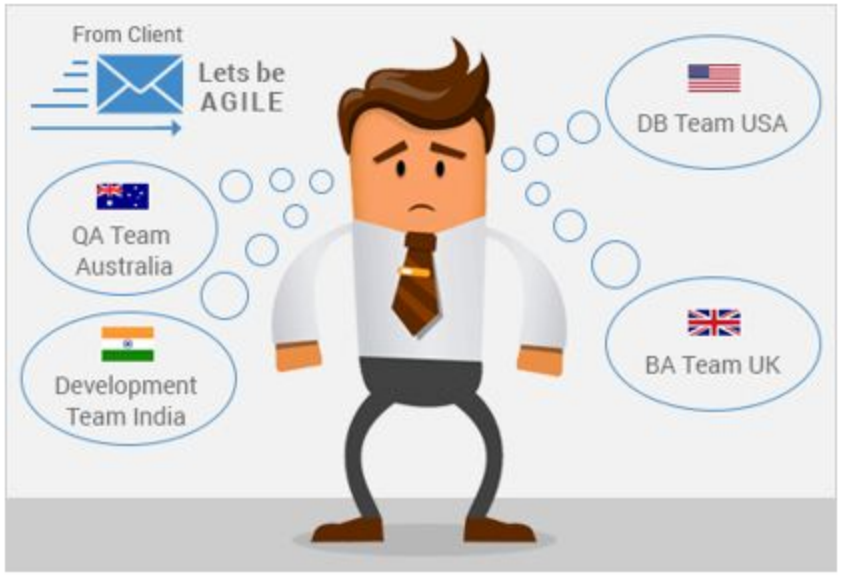
- Agile software process addresses **few assumptions**
  - **Difficulty in predicting changes** of requirements and customer priorities.
  - For many types of software; **design** and **construction** are **interleaved** (mixed).
  - **Analysis, design, construction** and **testing** are **not** as **predictable** as we might like.
- An agile **process** must be **adaptable**
- Requires **customer feedback**

Where agile methodology not work

Project plan & requirements are clear & unlikely to change

Unclear understanding of Agile Approach among Teams

Where agile methodology not work



Big Enterprises where team collaboration is tough

Agile Process Models

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Feature Driven Development (FDD)
- Crystal
- Agile Modelling (AM)

Extreme Programming (XP)

- The most widely used approach to agile software development
- A variant of XP called **Industrial XP (IXP)** has been proposed to target process for large organizations
- It uses **object oriented approach** as its preferred development model

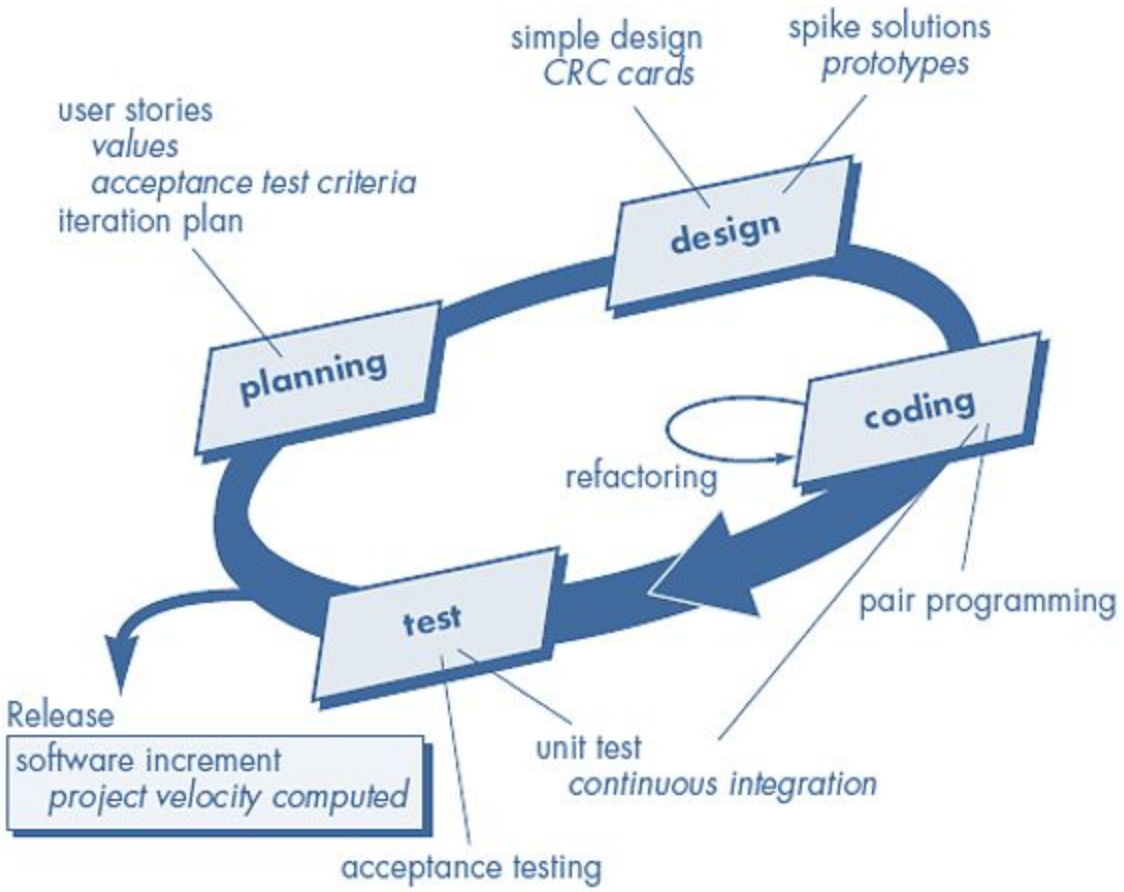
XP Values

- **Communication:** To achieve effective communication, it **emphasized close & informal (verbal) collaboration** between customers and developers
- **Simplicity:** It restricts developers to **design** for **immediate needs not** for **future needs**
- **Feedback:** It is derived **from** three sources the **implemented software**, the **customer** and **other software team members**, it uses **Unit testing** as primary testing
- **Courage:** It demands courage (discipline), there is often significant pressure to design for future requirements, XP team **must have the discipline (courage) to design for today**
- **Respect:** XP team **respect** among **members**

The XP Process

It considers four framework activities

1. Planning
2. Design
3. Coding
4. Testing



The XP Process cont.

Planning

- User Stories
  - **Customers assigns value** (priority)
  - **Developers assigns cost** (number of development weeks)
- Project velocity
  - Computed at the end of first release
  - **Number of stories implemented in first release**
  - Estimates for future release
  - **Guard against over-commitment**

Design

CRC card

Class Name	
Responsibilities	Collaborators

- **Keep-it-Simple** (Design of extra functionality is discouraged)
- **Preparation of CRC card** is work project
  - CRC cards identify and organize object oriented classes
- **Spike Solutions**
  - Operational prototype intended to clear confusion
- Refactoring
- Modify internals of code, No observable change

The XP Process cont.

Coding

- **Develops** a series of **Unit test** for stories included in current release
- **Complete code** perform **unit-test** to get immediate feedback
- XP recommend **pair-programming**, **"Two heads are better than one"**
- **Integrate code** with other team members, this **"continuous integration"** helps to avoid compatibility & interfacing problems, **"smoke testing"** environment to uncover errors early

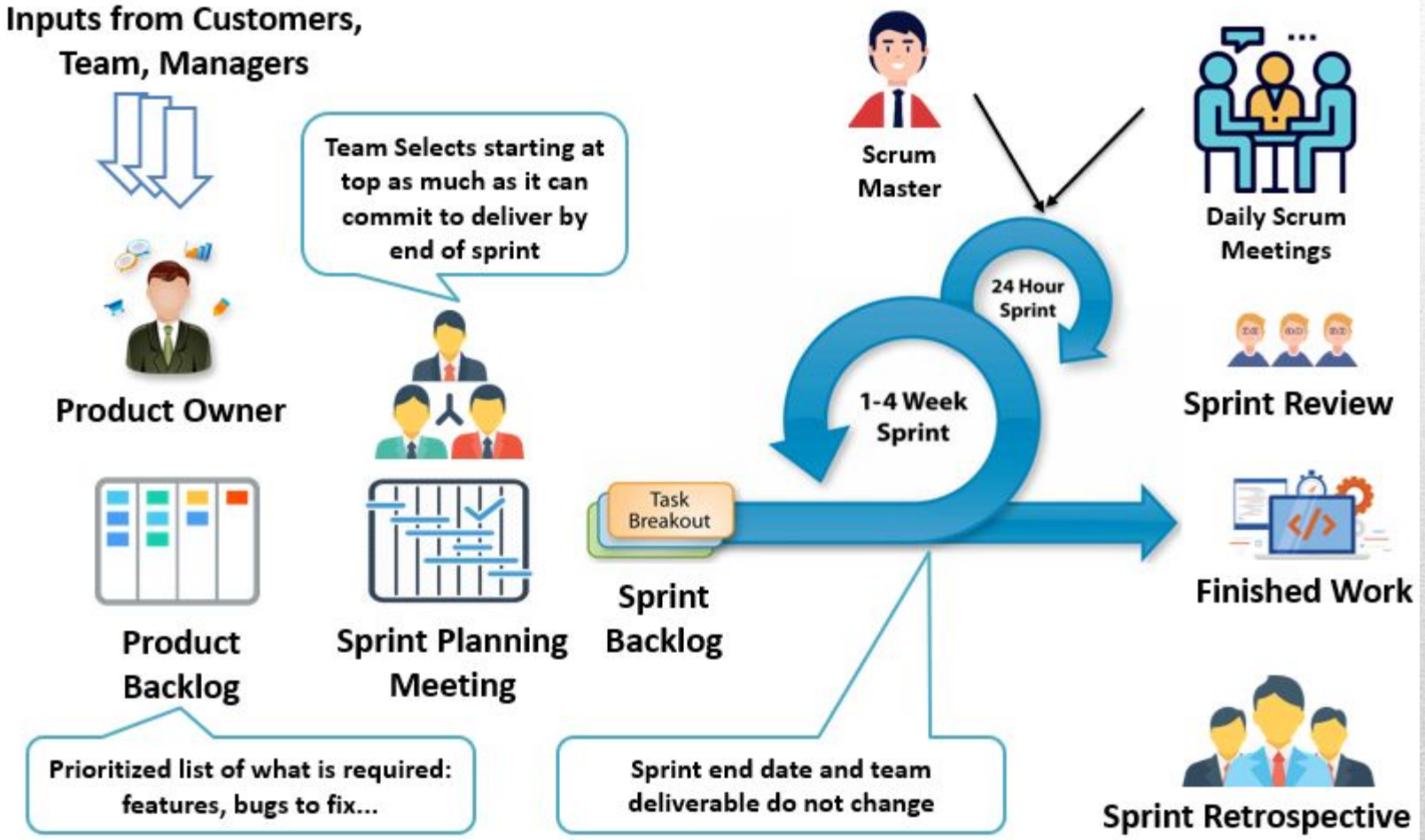
Testing

- **Unit test** by **developers** & fix small problems
- **Acceptance tests** - Specified by **customer**

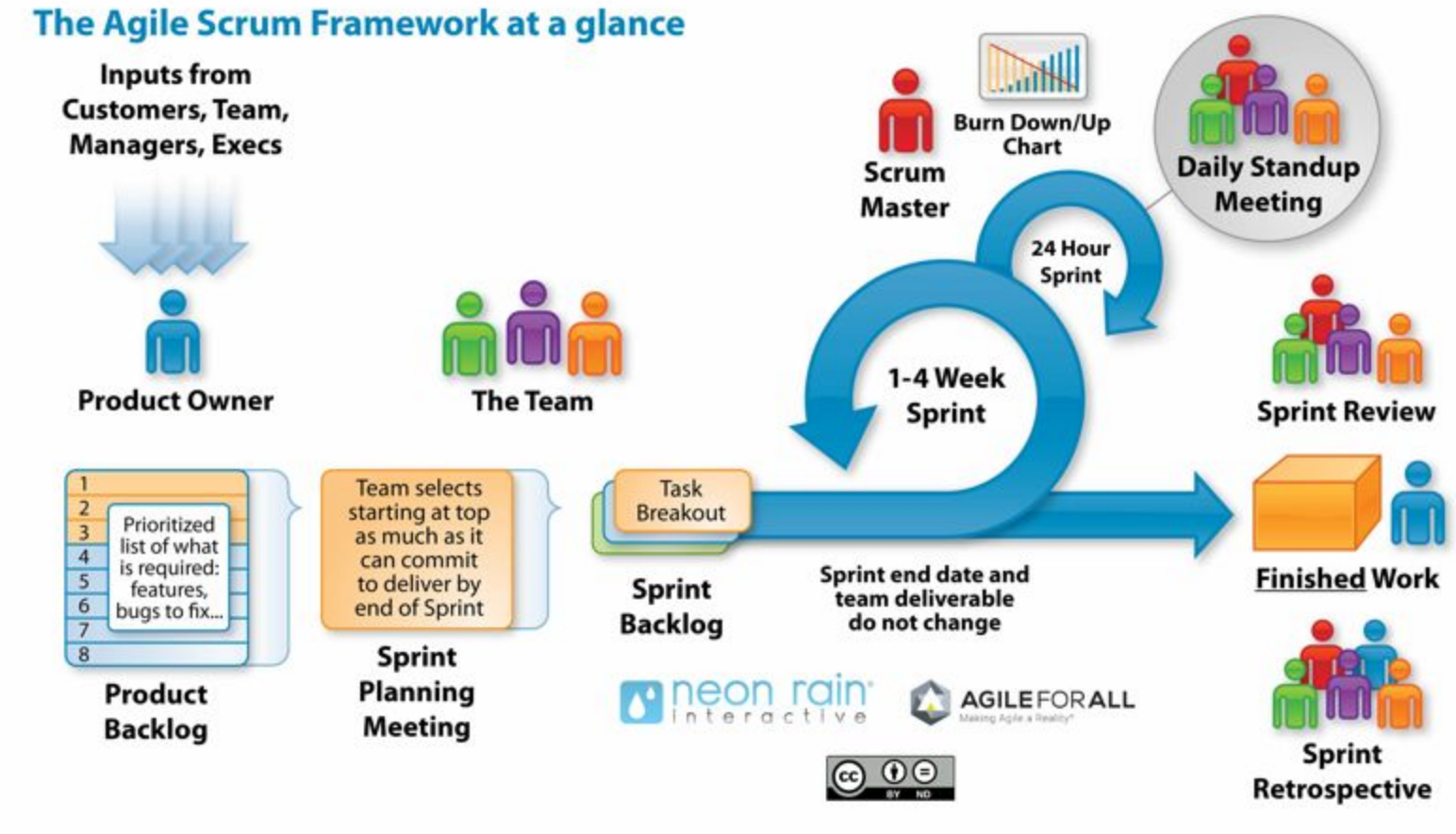
Scrum

- **Scrum** is an agile process model which is used for **developing** the **complex software** systems.
- It is a **lightweight process framework**.
- Lightweight means the **overhead of the process is kept as small** as possible in order to maximize the productivity.

Scrum framework at a glance



Scrum cont.


The Agile Scrum Framework at a glance
Inputs from Customers, Team, Managers, Execs
Product Owner
The Team
Scrum Master
Burn Down/Up Chart
Daily Standup Meeting
24 Hour Sprint
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Prioritized list of what is required: features, bugs to fix...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective
neon rain interactive
AGILEFORALL

Scrum cont.

1. **Backlog**
   - It is a **prioritized list of project requirements** or features that must be provided to the customer.
   - The **items can be included** in the backlog at **any time**.
   - The **product manager analyses** this **list** and **updates** the **priorities** as per the requirements.
2. **Sprint**
   - These are the **work units** that are needed **to achieve** the requirements mentioned in the backlogs.
   - Typically the sprints have **fixed duration** or time box (of **2 to 4 weeks, 30 days**).
   - **Change** are **not introduced** during the **sprint**.
   - Thus sprints allow the team **members** to **work** in **stable** and **short-term environment**.

Scrum cont.

3. Scrum Meetings

- There are **15 minutes daily meetings** to **report** the **completed** activities, **obstacles** and **plan** for **next** activities.
- Following are three questions that are mainly discussed during the meetings.
  1. **What** are the **tasks done** since **last meeting ?**
  2. **What** are the **issues** that team is **facing ?**
  3. **What** are the **next activities** that are **planned?**
- The **scrum master** leads the meeting and **analyses the response** of each team member.
- Scrum meeting **helps** the **team** to **uncover potential problems** as early as possible
- It leads to **"knowledge socialization"** & promotes **"self-organizing team structure"**

Scrum cont.

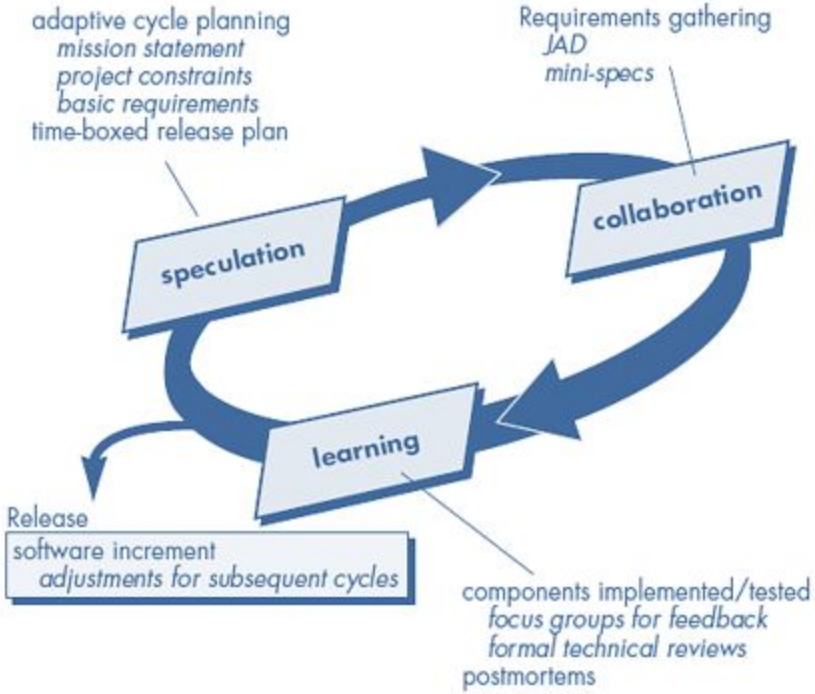
4. **Demo**
   - Deliver **software increment** to customer
   - Implemented functionalities are **demonstrated** to the customer

Adaptive Software development (ASD)

- This is a technique for building complex software systems using iterative approach.
- ASD focus on working in collaboration and team self-organization.

ASD incorporates three phases

1. Speculation
2. Collaboration
3. Learning



Speculation (ASD)

- The adaptive **cycle planning** is **conducted**.
- In this cycle planning mainly three types of information is used
  - Customer's **mission statement**
  - Project **constraints**
    - Delivery date, budgets etc…
  - **Basic requirements** of the project

Collaboration (ASD)

- In this, **collaboration** among the **members** of **development team** is a key factor.
- For **successful collaboration** and coordination it is necessary to have following **qualities** in every individual
  - **Assist each other** without resentment (offense)
  - Work **hard**
  - **Posses** the required **skill set**
  - **Communicate problems** and help each other
  - **Criticize without** any **hate**

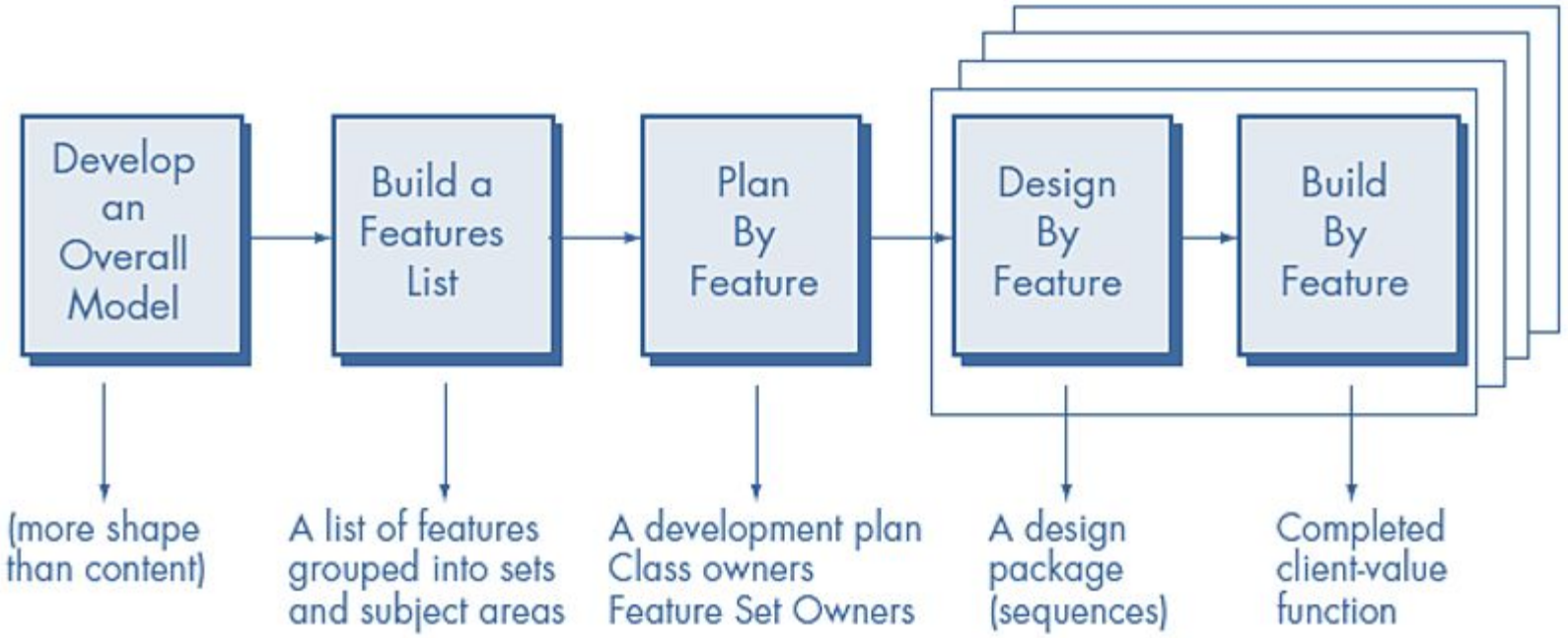
Learning (ASD)

- Emphasize is on **learning** new **skills** and techniques.
- There are three ways by which the team members learn
  - Focus groups
    - The **feedback** from the **end-users** is obtained.
  - Formal **technical review**
    - This review is conducted for better quality.
  - **Postmortems**
    - Team analyses its own performance and makes appropriate improvements.

DSDM

- **Dynamic Systems Development Methods (DSDM)**
- Various phases of this life cycle model
  - **Feasibility study**
    - By analysing the business requirements and constraints the viability of the application is determined
  - **Business study**
    - The functional and informational requirements are identified and then the business value of the application is determined
  - **Functional model iteration**
    - The incremental approach is adopted for development
  - **Design and build iteration**
    - If possible design and build activities can be carried out in parallel
  - **Implementation**
    - The software increment is placed in the working environment

Feature Driven Development (FDD)



FDD cont.

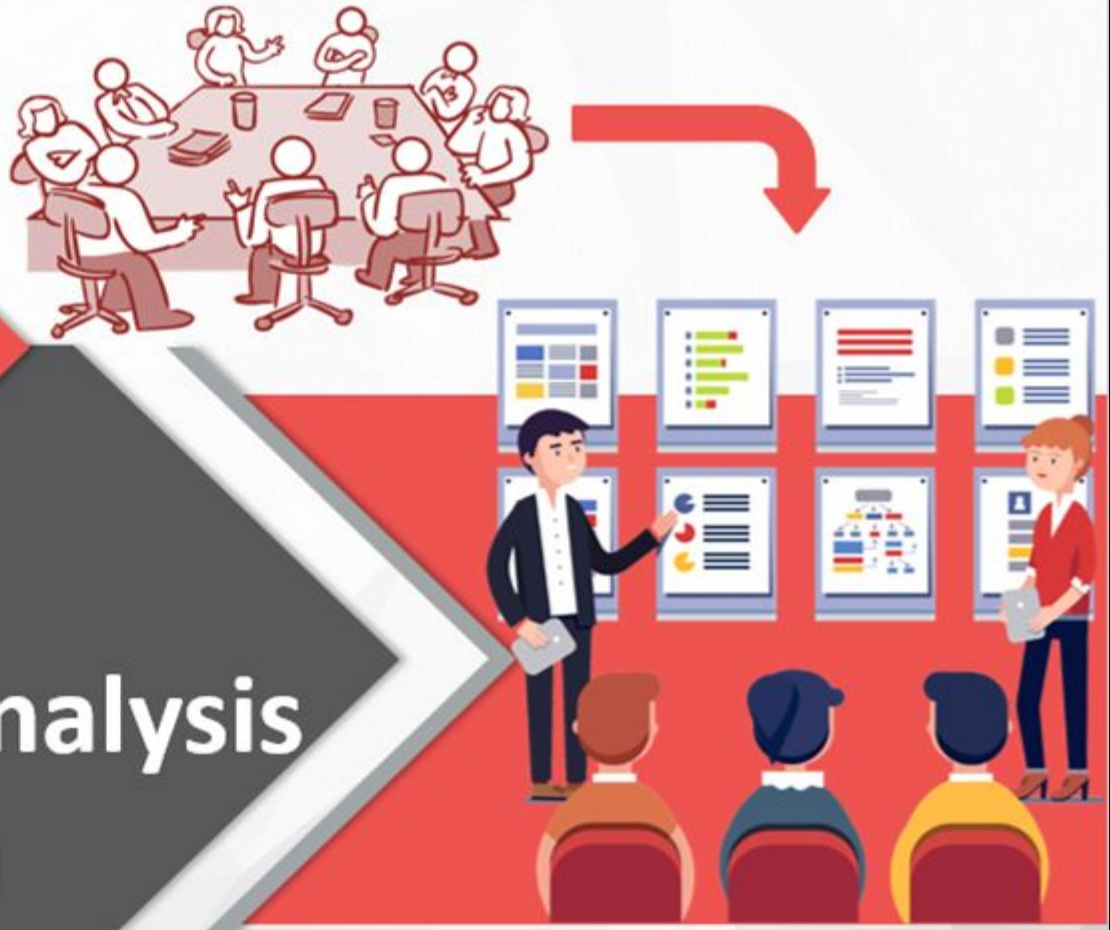
- It is practical process model for **object oriented software engineering**.
- In FDD, the feature means client valued function.
- Various **phases** in the FDD life cycle
  1. **Develop overall model**
     - The high-level walkthrough of scope and detailed domain walkthrough are conducted to create overall models.
  2. **Build feature list**
     - List of features is created and expressed in the following form
       - **<action>** the **<result>** **<by for of to>** a(n) **<object>**
       - **For Ex. "Display product-specifications of the product"**

FDD cont.

3. **Plan by feature**
   - After completing the feature list the development plan is created
4. **Design by feature**
   - For each feature the sequence diagram is created
5. **Build by feature**
   - Finally the class owner develop the actual code for their classes

Software Engineering

Requirement Analysis & Specification



Software Engineering: A Practitioner's Approach, 8/e **by Roger S. Pressman**

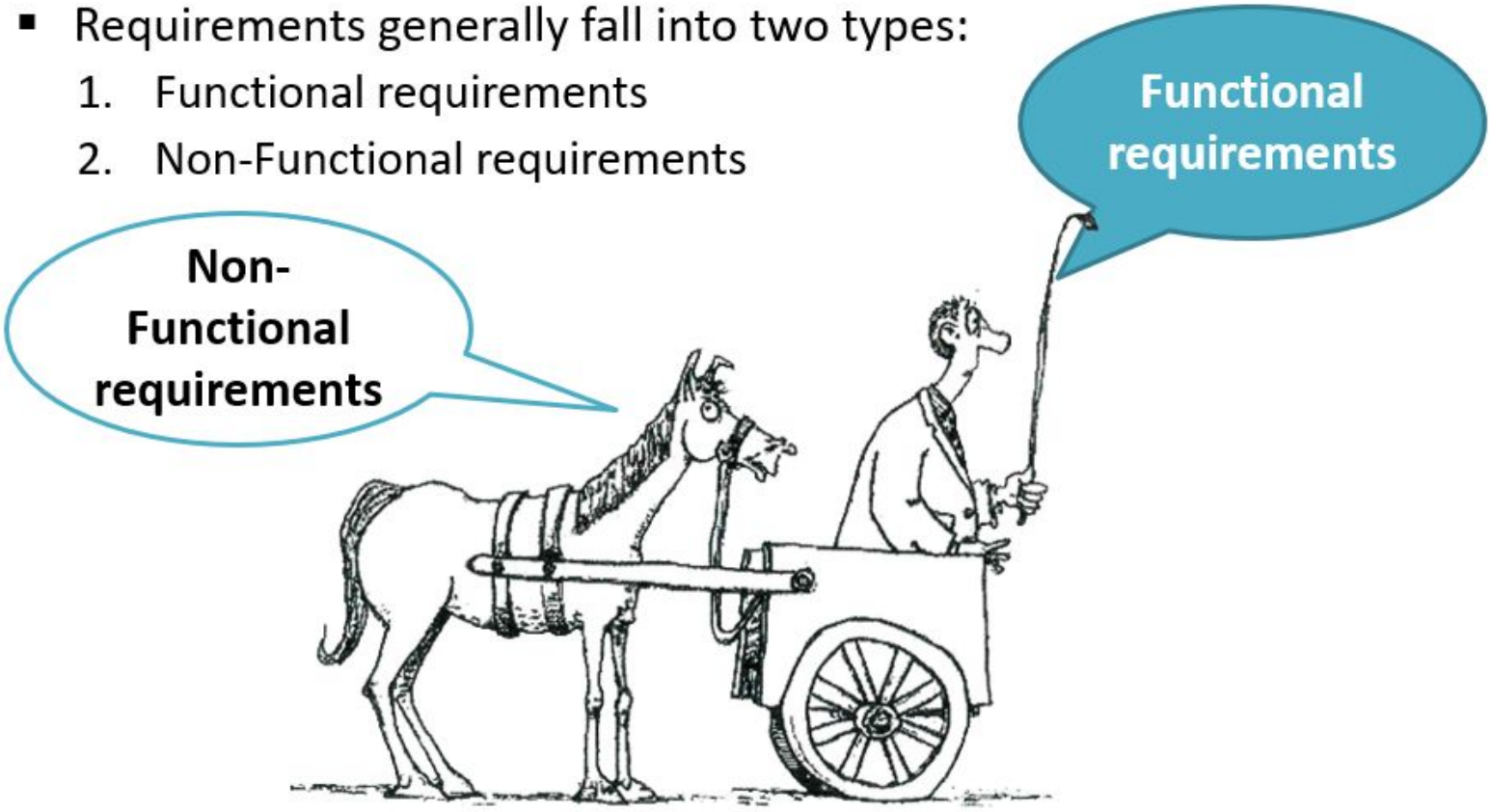
Software Engineering 10/e **By Ian Sommerville**

Requirement Engineering

- **Tasks** and **techniques** that lead to an **understanding** of **requirements** is called **requirement engineering**.
- Requirement engineering **provides** the appropriate **mechanism** for **understanding**
  - What customer wants
  - Analyzing needs
  - Assessing feasibility
  - Negotiating a reasonable solution
  - Specifying solution unambiguously
  - Validating the specification
  - Managing requirements

Functional & Non-Functional requirements

- Requirements generally fall into two types:
  1. Functional requirements
  2. Non-Functional requirements



Don't put what you want to do - before how you need to do it

Functional requirements

Any requirement which specifies what the system should do.

A functional requirement will describe a particular behaviour of function of the system when certain conditions are met, for example: "Send email when a new customer signs up" or "Open a new account".

Typical functional requirements include:

- Business Rules
- Transaction corrections, adjustments and cancellations
- Administrative functions
- Authentication
- Authorization levels
- Audit Tracking
- External Interfaces
- Reporting Requirements
- Historical Data
- Legal or Regulatory Requirements

Non-Functional requirements

Any requirement which specifies how the system performs a certain function.

A non-functional requirement will describe how a system should behave and what limits there are on its functionality.

Typical Non-Functional requirements include:

• Response time	• Availability	• Regulatory
• Throughput	• Reliability	• Manageability
• Utilization	• Recoverability	• Environmental
• Static volumetric	• Maintainability	• Data Integrity
• Scalability	• Serviceability	• Usability
• Capacity	• Security	• Interoperability

Library Management System

Function Requirements

- **Add Article:** New entries must be entered in database
- **Update Article:** Any changes in articles should be updated in case of update
- **Delete Article:** Wrong entry must be removed from system
- **Inquiry Members:** Inquiry all current enrolled members to view their details
- **Inquiry Issuance:** Inquiry all database articles
- **Check out Article:** To issue any article must be checked out
- **Check In article:** After receiving any article system will reenter article by Checking
- **Inquiry waiting for approvals:** Librarian will generates all newly application which is in waiting list
- **Reserve Article:** This use case is used to reserve any book with the name of librarian, it can be pledged
- **Set user Permission:** From this user case Librarian can give permission categorically, alsoenabling/disabling of user permission can be set through this use case

Library Management System

Non-Function Requirements

- **Safety Requirements:** The database may get crashed at any certain time due to virus or operating system failure. Therefore, it is required to take the database backup
- **Security Requirements:** We are going to develop a secured database for the university. There are different categories of users namely teaching staff, administrator, library staff ,students etc., Depending upon the category of user the access rights are decided. It means if the user is an administrator then he can be able to modify the data, delete, append etc., all other users other than library staff only have the rights to retrieve the information about database.
- **Software Constraints:** The development of the system will be constrained by the availability of required software such as database and development tools. The availability of these tools will be governed by

Requirements Engineering Tasks

1 Inception

- Roughly define scope
- A basic understanding of a problem, people who want a solution, the nature of solution desired

2 Elicitation (Requirement Gathering)

- Define requirements
- The practice of collecting the requirements of a system from users, customers and other stakeholders

Requirements Engineering Tasks cont.

3 Elaboration

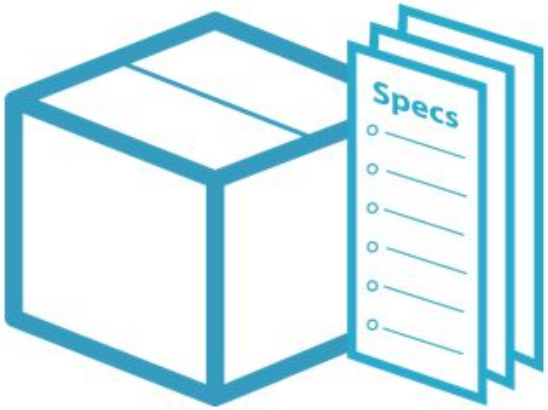
- Further define requirements
- Expand and refine requirements obtained from inception & elicitation
- Creation of User scenarios, extract analysis class and business domain entities

4 Negotiation

- Reconcile conflicts
- Agree on a deliverable system that is realistic for developers and customers

Requirements Engineering Tasks cont.

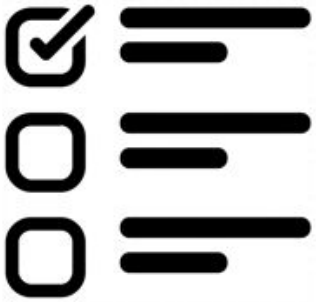
5 Specification



- Create analysis model
- It may be written document, set of graphical models, formal mathematical model, collection of user scenarios, prototype or collection of these
- **SRS (Software Requirement Specification)** is a document that is created when a detailed description of all aspects of software to build must be specified before starting of project

Requirements Engineering Tasks cont.

6 Validation



- Ensure quality of requirements
- Review the requirements specification for errors, ambiguities, omissions (absence) and conflicts

7 Requirements Management

- It is a set of activities to identify, control & trace requirements & changes to requirements (Umbrella Activities) at any time as the project proceeds.

Elicitation is the Hardest Part!

- **Problems of scope**
  - System boundaries are ill-defined
  - Customers will provide irrelevant information
- **Problems of understanding**
  - Customers never know exactly what they want
  - Customers don't understand capabilities and limitations
  - Customers have trouble fully communicating needs
- **Problems of volatility**
  - Requirements always change

Project Inception

- During the **initial project meetings**, the following tasks should be accomplished
  - **Identify the project stakeholders**
    - These are the folks we should be talking to
  - **Recognize multiple viewpoints**
    - Stakeholders may have different (and conflicting) requirements
  - **Work toward collaboration**
    - It's all about reconciling conflict
  - **Ask the first questions**
    - Who? What are the benefits? Another source?
    - What is the problem? What defines success? Other constraints?
    - Am I doing my job right?

Collaborative Elicitation

- One-on-one Q&A sessions rarely succeed in practice; collaborative strategies are more practical

Elicitation work products

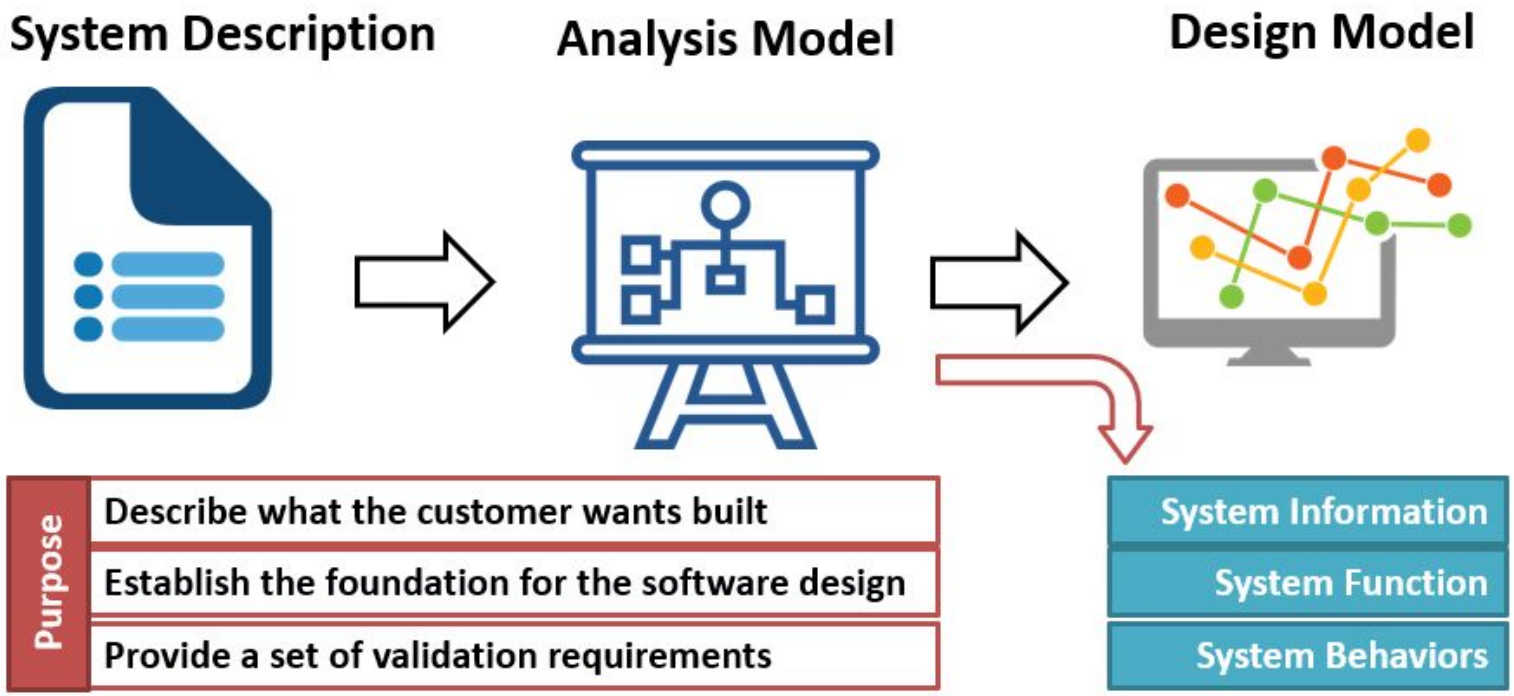
- Collaborative elicitation should result in several **work products**

  - A bounded statement of scope
  - A list of stakeholders
  - A description of the technical environment
  - A list of requirements and constraints
  - Any prototypes developed
  - A set of use cases
    - Characterize how users will interact with the system
    - Use cases tie functional requirements together

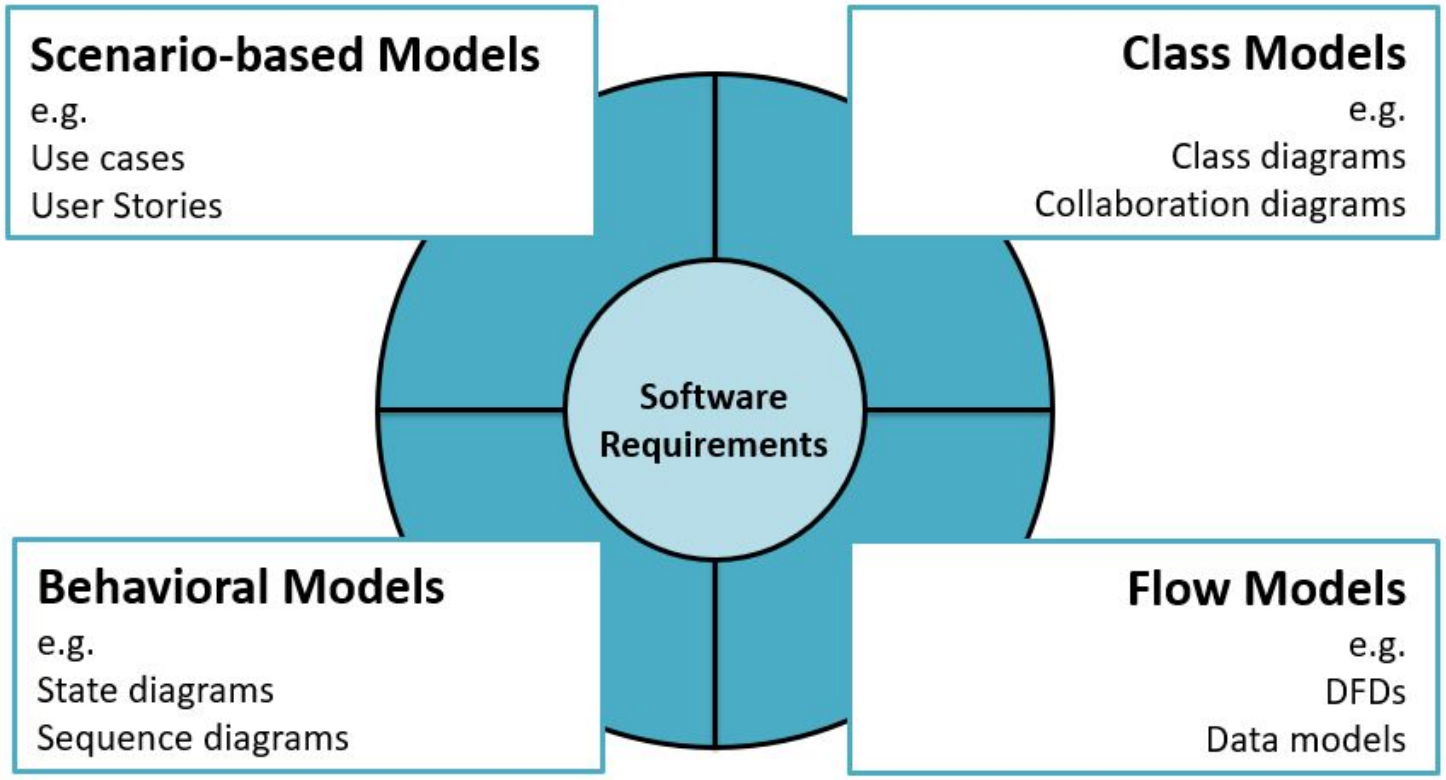
Quality Function Deployment (QFD)

- This is a technique that **translates** the **needs** of the **customer** into **technical requirements** for software
- It **emphasizes** an understanding of **what is valuable to the customer** and then deploys these values throughout the engineering process through functions, information, and tasks
- It **identifies** three types of **requirements**
  - **Normal requirements:** These requirements are the objectives and goals stated for a product or system during meetings with the customer
  - **Expected requirements:** These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them
  - **Exciting requirements:** These requirements are for features that go beyond the customer's expectations and prove to be very satisfying when present

The requirement analysis model

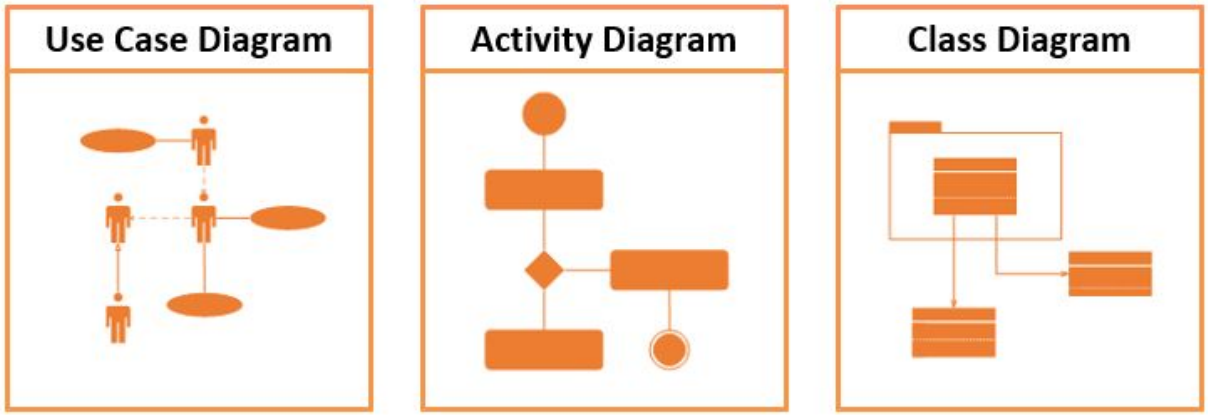


Elements of the Requirements Model



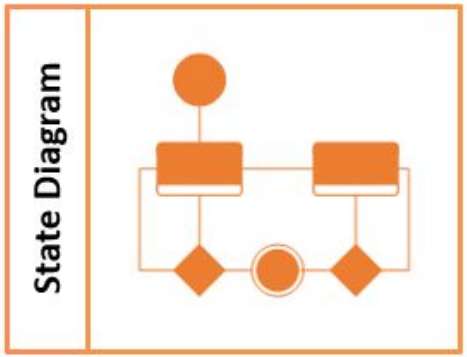
Elements of the Requirements Model

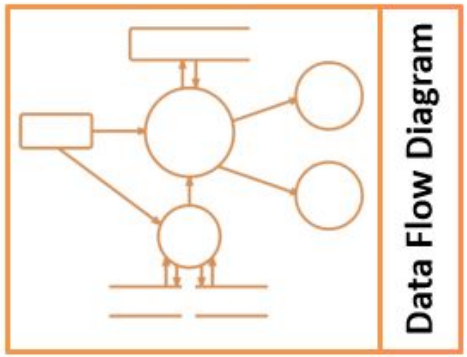
- **Scenario-based elements**
  - Describe the system from the user's point of view using scenarios that are depicted (stated) in use cases and activity diagrams
- **Class-based elements**
  - Identify the domain classes for the objects manipulated by the actors, the attributes of these classes, and how they interact with one another; which utilize class diagrams to do this.



Elements of the Requirements Model

- **Behavioral elements**
  - Use state diagrams to represent the state of the system, the events that cause the system to change state, and the actions that are taken as a result of a particular event.
  - This can also be applied to each class in the system.
- **Flow-oriented elements**
  - Use data flow diagrams to show the input data that comes into a system, what functions are applied to that data to do transformations, and what resulting output data are produced.





Analysis Modeling Approaches

Structured Analysis

- Models data elements
  - Attributes
  - Relationships
- Models processes that transform data

Object Oriented Analysis

- Models analysis classes
  - Data
  - Processes
- Models class collaborations

Techniques from both approaches are typically used in practice.

Software Requirements Specification

- Software Requirement Specification (SRS) is a document that completely describes what the proposed software should do without describing how software will do it.
- It contains:
  - a complete information description
  - a detailed functional description
  - a representation of system behaviour
  - an indication of performance requirements and design constraints
  - appropriate validation criteria
  - other information suitable to requirements
- SRS is also helping the clients to understand their own needs.

Characteristics of a Good SRS

- SRS should be accurate, complete, efficient, and of high quality
  - so that it does not affect the entire project plan.
- An SRS is said to be of high quality when the developer and user easily understand the prepared document.
- Characteristics of a Good SRS:
  - **Correct**
    - SRS is correct when all user requirements are stated in the requirements document.
    - Note that there is no specified tool or procedure to assure the correctness of SRS.
  - **Unambiguous**
    - SRS is unambiguous when every stated requirement has only one interpretation.

Characteristics of a Good SRS (Cont…)

- **Complete**
  - SRS is complete when the requirements clearly define what the software is required to do.
- **Ranked for Importance/Stability**
  - All requirements are not equally important, hence each requirement is identified to make differences among other requirements.
  - Stability implies the probability of changes in the requirement in future.
- **Modifiable**
  - The requirements of the user can change, hence requirements document should be created in such a manner that those changes can be modified easily.
- **Traceable**
  - SRS is traceable when the source of each requirement is clear and facilitates the reference of each requirement in future.

Characteristics of a Good SRS (Cont…)

- **Verifiable**
  - SRS is verifiable when the specified requirements can be verified with a cost-effective process to check whether the final software meets those requirements.
- **Consistent**
  - SRS is consistent when the subsets of individual requirements defined do not conflict with each other.

Standard Template for writing SRS

- **Front Page**

  Software Requirements Specification

  for

  <Project>

  Version <no.>

  Prepared by <author>

  <organization>

  <date created>
- **Table of Contents**
- **Revision History**

Standard Template for writing SRS (Cont…)

1. Introduction

- 1.1 Purpose
- 1.2 Document Conventions
- 1.3 Intended Audience and Reading Suggestions
- 1.4 Project Scope
- 1.5 References

2. Overall Description

- 2.1 Product Perspective
- 2.2 Product Features
- 2.3 User Classes and Characteristics
- 2.4 Operating Environment
- 2.5 Design and Implementation Constraints

Standard Template for writing SRS (Cont…)

2.6 User Documentation

2.7 Assumptions and Dependencies

3. System Features

3.1 System Feature 1

3.2 System Feature 2 (and so on)

4. External Interface Requirements

4.1 User Interfaces

4.2 Hardware Interfaces

4.3 Software Interfaces

4.4 Communications Interfaces

Standard Template for writing SRS (Cont…)

5. Other Nonfunctional Requirements

- 5.1 Performance Requirements
- 5.2 Safety Requirements
- 5.3 Security Requirements
- 5.4 Software Quality Attributes

6. Other Requirements

Appendix A: Glossary

Appendix B: Analysis Models

Appendix C: Issues List

Problems Without SRS

- Without developing the SRS document, the system would not be properly implemented according to customer needs.
- Software developers would not know whether what they are developing is what exactly is required by the customer.
- Without SRS, it will be very difficult for the maintenance engineers to understand the functionality of the system.
- It will be very difficult for user document writers to write the users' manuals properly without understanding the SRS.

Summary

- Requirements Engineering
  - Requirements Engineering Tasks
  - Eliciting Requirements
  - Collaborative Requirements Gathering
  - Quality Function Deployment
  - Usage Scenarios
  - Elicitation Work Products
- Requirements Analysis Model
  - Use-Case Diagram
  - Class Diagram
  - State Diagram
  - Data Flow Diagram
- Negotiating Requirements
- Functional and non-functional requirements
- SRS

Next…Class

"Unit 2: Software Modeling"