MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

CET2013B: Software Engineering and Testing

Cyber Security and Forensics

S.Y. Semester IV

2022-23

CET2013B: Software Engineering and Testing

Examination Scheme :

Continuous Assessment: 60 Marks End Semester Examination :40 Marks Credit:3+1=4

- **Course Objectives:**

  1.Knowledge:

  a. To provide the knowledge of Software Design Methodologies strategies.

  b. To understand how testing methods can be used as an effective tool in quality assurance of software.

  2. Skills:

  a. To understand different design methodologies and implement for the system.

  b. To provide skills to design test case plan for testing software.

  3. Attitude:

  a. To acquaint with the different testing methodologies and tools.

  b. To analyze and design any real time system with appropriate test cases.

- **Course Outcomes:**

1. To transform requirement document into an appropriate design.
2. Apply appropriate UML diagrams and notations to design the product.
3. Identify the bugs and create various test cases based on the application.
4. To understand and apply testing techniques in the software development.

Guidelines for SET

Class Continuous Assessment (CCA): 60

Lab Continuous Assessment (LCA): 50

Term End Exam: 40

CCA Marks

Exam	Weightage	Marks
Mid-Term	**33%**	**20**
Presentation	**25%**	**15**
Theory Assignment	**25%**	**15**
Case Study/Active Learning	**17%**	**10**

Criteria	Sub Criteria	Marks
Content	Correct Content	**4**
	Average Content	**3**
	Insufficient Content	**2-1**
Submission	On time Submission	**4**
	Delay by 1 Day	**3**
	Delay by 2 Day	**2**
	After 2 Days till Midterm	**1**
Neatness	Good	**2**
	Average	**1**

LCA Marks

LCA Marks Distribution

Exam	Weightage	Marks
Laboratory	**60%**	**30**
Oral	**40%**	**20**

Practical Rules

Criteria	Sub Criteria	Marks
Understanding and Performance	Problem Statement understood and analyzed	**4-5**
	Problem Statement moderately understood and moderately analyzed	**2-3**
	Program Statement not understood and not analyzed	**1**
Completion	On time completion	**3**
	Partial completion	**2**
	Delay in completion	**1**
Innovation/Ethics	Extension/Innovation	**2**
	Copied from others but understood	**1**

Unit 1. Software Engineering Fundamentals

- Introduction to Software Engineering
- Importance of Software Engineering
- Software Engineering Practice , Software Myths
- Software Process, Software Perspective and Specialized Process Models.

- Case Studies
- Software Requirements, User requirements, System requirements
- Software Requirements Specification
- Functional Specification and Non-functional specifications

Disclaimer:

a. Information included in this slides came from multiple sources. We have tried our best to cite the sources. Please refer to the References to learn about the sources, when applicable.

b. The slides should be used only for academic purposes (e.g., in teaching a class), and should not be used for commercial purposes.

What is Software?

Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information and (3) documentation that describes the operation and use of the programs.

The IEEE definition of Software Engineering

Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).

Without software, most computers would be useless. For example, without your internet browser software, you could not surf the Internet or read this page. Without an operating system, the browser could not run on your computer.

Software

How do you get software?

- Software can be purchased at a retail computer store or online and come in a box containing all the disks (floppy diskette, CD, DVD, or Blu-ray), manuals, warranty, and other documentation.
- Software can also be downloaded to a computer over the Internet. Once downloaded, setup files are run to start the installation process on your computer.

Free software : There are also a lot of free software programs available that are separated into different categories.

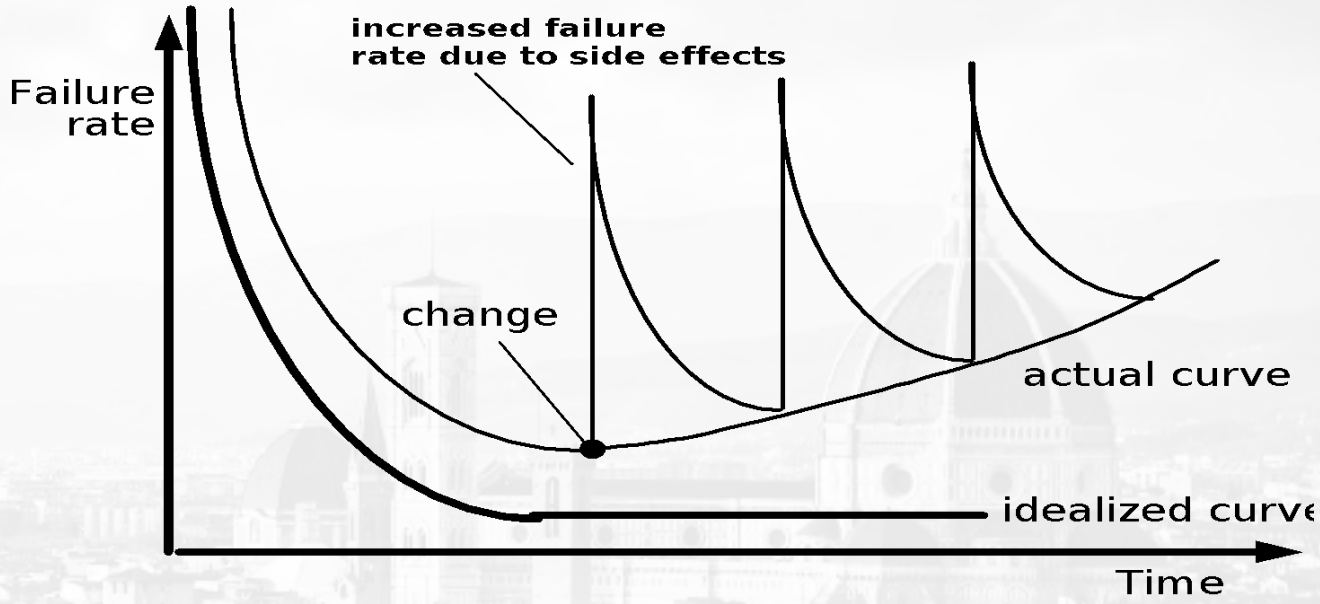
- **Shareware or trial software** is software that gives you a few days to try the software before you have to buy the program. After the trial time expires, you'll be asked to enter a code or register the product before you can continue to use it.
- **Freeware** is completely free software that never requires payment, as long as it is not modified.
- **Open source software** is similar to freeware. Not only is the program given away for free, but the source code used to make the program is as well, allowing anyone to modify the program or view how it was created.

Examples

Software	Examples
Antivirus	AVG, Housecall, McAfee, and Norton.
Audio / Music program	iTunes and WinAmp.
Database	Access, MySQL, and SQL.
Device drivers	Computer drivers.
E-mail	Outlook and Thunderbird.
Game	Madden NFL football, Quake, and World of Warcraft.
Internet browser	Firefox, Google Chrome, and Internet Explorer.
Movie player	VLC and Windows Media Player.
Operating system	Android, iOS, Linux, macOS, and Windows.

Importance of Software

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn't "wear out."
- Although the industry is moving towards component-based construction, most software continues to be custom-built.



Software Applications

1. **System software:** System software serves as the interface between the hardware and the end users.

Some examples of system software are Operating System, Compilers, Interpreter, Assemblers, etc.

2. **Application software:** e.g. Payroll Software, Student Record Software, Inventory Management Software, Income Tax Software, Railways Reservation Software, Microsoft Office Suite Software, Microsoft Word.

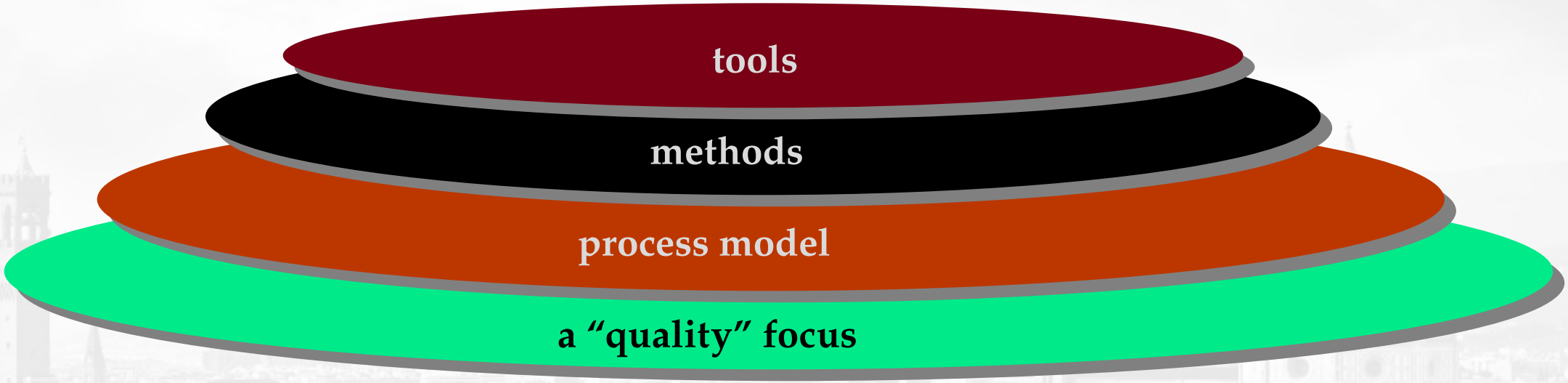
3. **Engineering/scientific software:** e.g. CAD/CAM, automatic stress analysis, nuclear biology, Space shuttle.

4.**Embedded software:** Embedded software is a piece of software that is **embedded in hardware or non-PC devices**. It is written specifically for the particular hardware that it runs on and usually has **processing and memory constraints** because of the device's limited computing capabilities**.**

5.**WebApps (Web applications):** client–server computer program which runs in a web browser. Common web applications include webmail, online retail sales, and online auction.

6. **AI software:** e.g. Robotics, Expert system, pattern recognition, image, voice, ANN.

Software :A Layered Technology



Software Engineering

Process Framework

- A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of size or complexity.
- It also includes a set of umbrella activities that are applicable across the entire software process.
- Each framework activity is populated by a set of software engineering actions – a collection of related tasks that produces a major software engineering work product (e.g. design is a software engineering action).
- Each action is populated with individual work tasks that accomplish some part of the work implied by the action.

A Process Framework

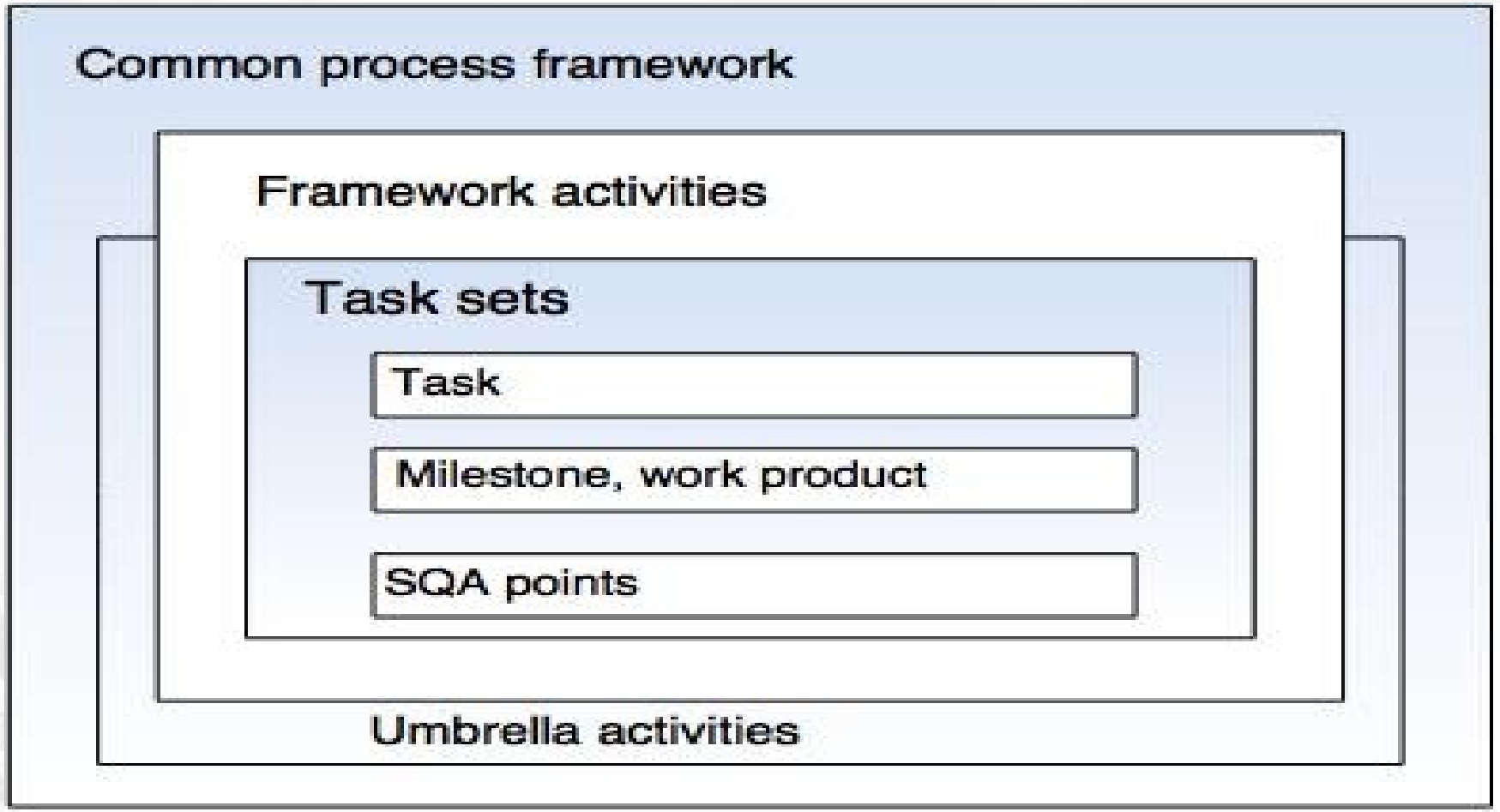


Fig.- A software process framework

Framework Activities

- Communication
- Planning
- Modeling
    - Analysis of requirements
    - Design
- Construction
    - Code generation
    - Testing
- Deployment

Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

Identifying a Task Set

- Before you can proceed with the process model, a key question: what **actions** are appropriate for a framework activity given the nature of the problem, the characteristics of the people and the stakeholders?

- **A task set defines the actual work to be done** to accomplish the objectives of a software engineering action.
  - A list of the task to be accomplished
  - A list of the work products to be produced
  - A list of the quality assurance filters to be applied

Identifying a Task Set

- For example, a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone all with the stakeholder. The work tasks of this action are:

1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval.

Software Myths

- All people who come into contact with software may suffer from various myths associated with developing and using software. Here are a few common ones.

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth, but …
- Invariably lead to bad decisions, therefore …
- Insist on reality as you navigate your way through software engineering

Management Myths

- **Myth:** The members of an organization can acquire all-the information, they require from a manual, which contains standards, procedures, and principles.
- **Reality**:
  1. Standards are often incomplete, inadaptable, and outdated.
  2. Developers are often **unaware** of all the established standards.
  3. Developers rarely follow all the known standards because not all the standards tend to decrease the delivery time of software while maintaining its quality.
- **Myth:** If the project is behind schedule, increasing the number of programmers can reduce the time gap.
- **Reality:**
  1. Adding more manpower to project, that is behind schedule, further delays the project.

  Why?

User Myths

- **Myth:** If the project is outsourced to a third party, the management can relax and let the other firm develop software for them.
- **Reality:** Outsourcing software to a third party does not help the organization, which is incompetent in managing and controlling the software project internally. The organization invariably suffers when it out sources the software project.
- **Myth:** Brief requirement stated in the initial process is enough to start development; detailed requirements can be added at the later stages.
- **Reality:**

  1.Starting development with incomplete and ambiguous requirements often lead to software failure. Instead, a complete and formal description of requirements is essential before starting development.

  2.Adding requirements at a later stage often requires repeating the entire development process.

User Myths

- **Myth:** Software is flexible; hence software requirement changes can be added during any phase of the development process.
- **Reality**: Incorporating change requests earlier in the development process costs lesser than those that occurs at later stages. This is because incorporating changes later may require **redesigning** and **extra resources**.

Developer Myths

- **Myth:** Software development is considered complete when the code is delivered.
- **Reality:** 50% to 70% of all the efforts are exhausted after the software is delivered to the user.
- **Myth:** The success of a software project depends on the quality of the product produced.
- **Reality:** The quality of programs is not the only factor that makes the project successful instead the documentation and software configuration also play a crucial role.
- **Myth:** Software engineering requires unnecessary documentation, which slows down the project.
- **Reality**: Software engineering is about creating quality at every level of the software project. Proper documentation enhances quality which results in reducing the amount of rework.

Developer Myths

- **Myth:** The only product that is delivered after the completion of a project is the working program(s).
- **Reality:** The deliverables of a successful project includes not only the working program but also the documentation to guide the users for using the software.
- **Myth:** Software quality can be assessed only after the program is executed.
- **Reality:** The quality of software can be measured during any phase of development process by applying some quality assurance mechanism. One such mechanism is formal technical review that can be effectively used during each phase of development to uncover certain errors.

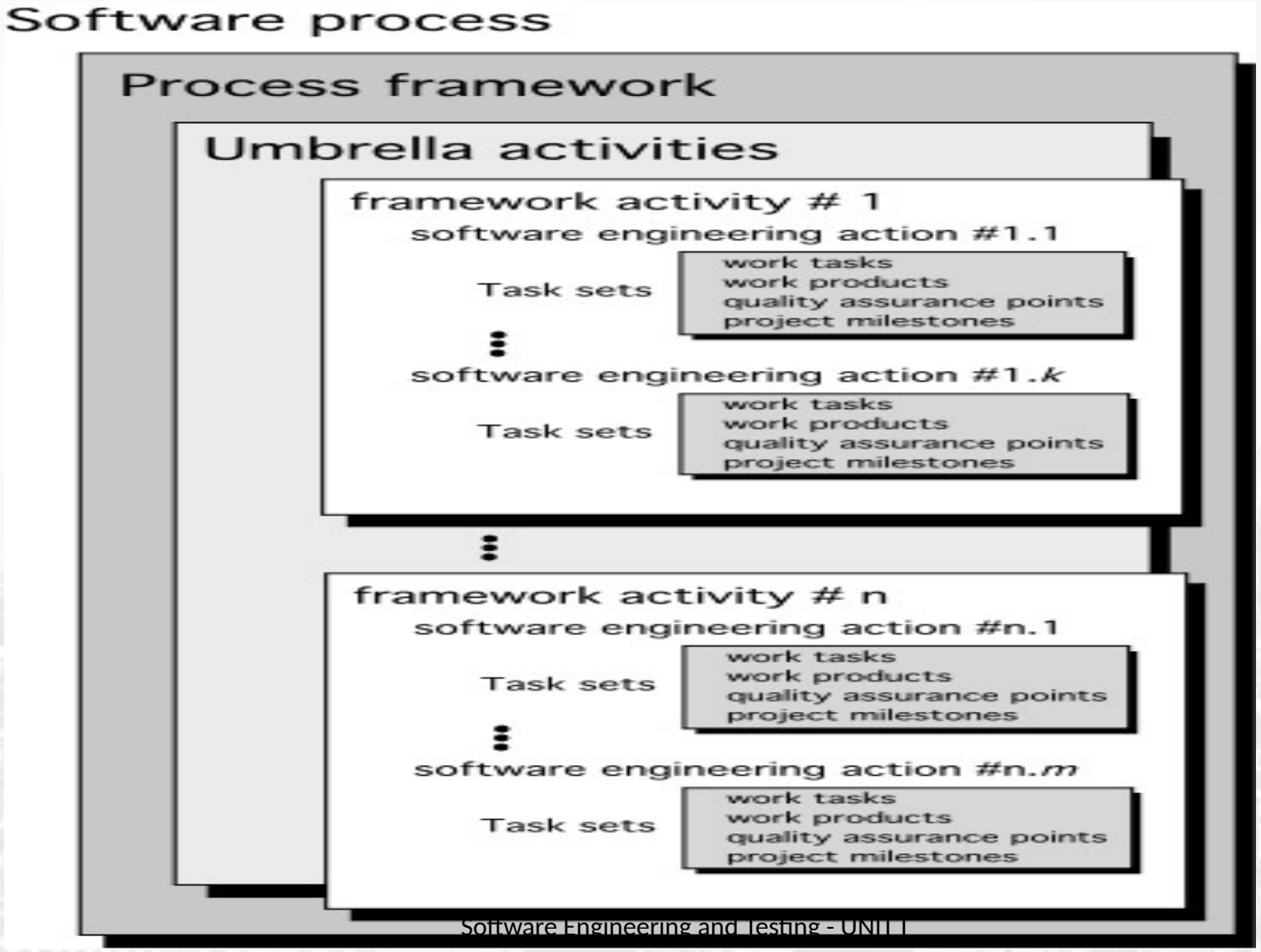
The Software Engineering Process

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - communication
  - planning
  - modeling
  - construction and
  - deployment
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Definition of Software Engineering Process

- A **framework** for the activities, actions, and tasks that are required to build high-quality software.
- Software Process defines the approach that is taken as software is engineered.
- Is not equal to software engineering-which also encompasses **technologies** that populate the process– technical methods and automated tools.

A Generic Process Model

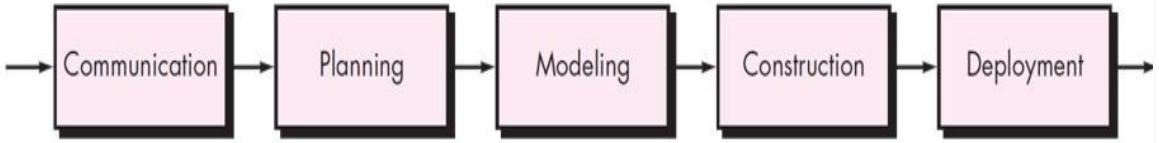


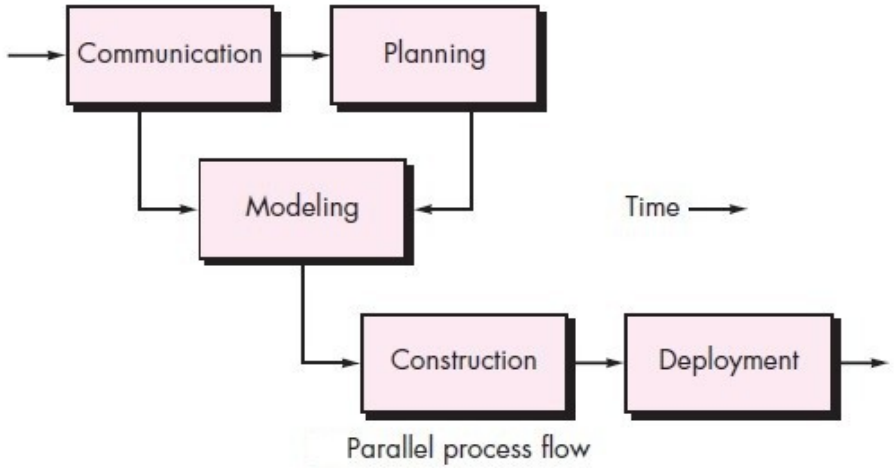
A Generic Process Model

- A generic process framework for software engineering defines **five framework activities-communication, planning, modeling, construction, and deployment**.
- In addition, a set of umbrella activities- project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.
- Next question is: how the framework activities, the actions and tasks that occur within each activity are organized **with respect to sequence and time**?

Process Flow

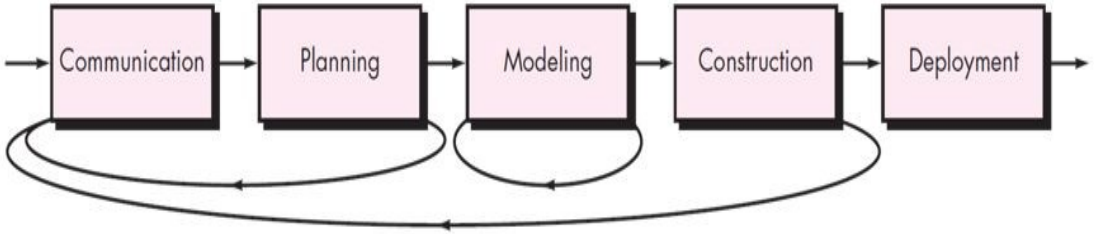
- Linear process flow

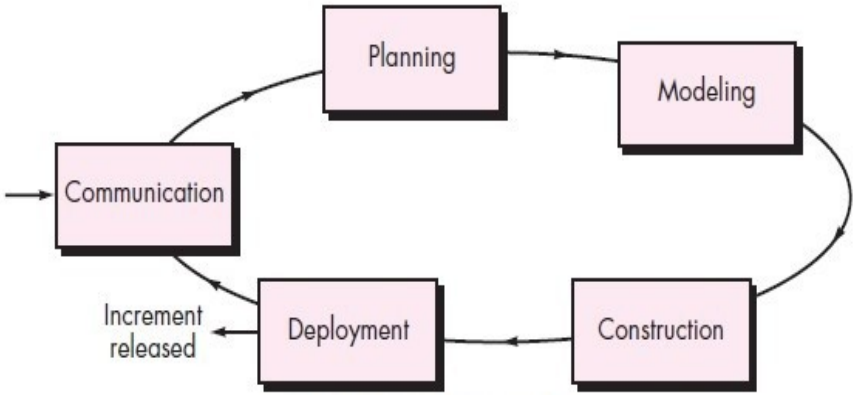




Parallel process flow

- Iterative process flow





Evolutionary process flow

Process Flow

- Linear process flow executes each of the five activities in sequence.
- An iterative process flow repeats one or more of the activities before proceeding to the next.
- An evolutionary process flow executes the activities in a circular manner. Each circuit leads to a more complete version of the software.
- A parallel process flow executes one or more activities simultaneously with other activities (modeling for one aspect of the software in parallel with construction of another aspect of the software).

Adapting a Process Model

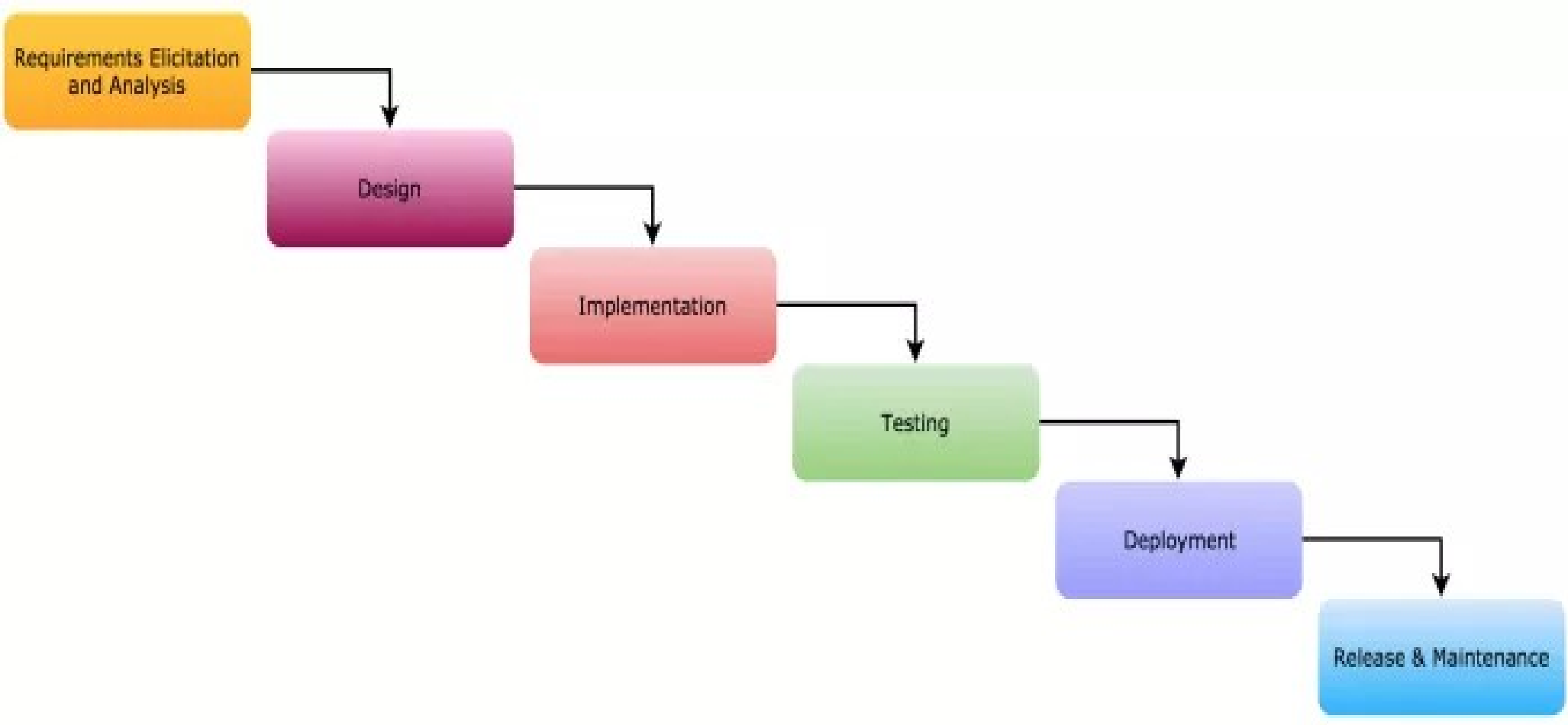
Depends on :

- overall flow of **activities, actions, and tasks** and the interdependencies among them
- the degree to which actions and tasks are defined within each framework activity
- the degree to which work products are identified and required
- the manner which quality assurance activities are applied
- the manner in which project tracking and control activities are applied
- the overall degree of detail and rigor with which the process is described
- the degree to which customer and other stakeholders are involved with the project
- the level of autonomy given to the software team
- the degree to which team organization and roles are prescribed

Software Process Models

- **WATERFALL MODEL**
- **INCREMENTAL PROCESS MODEL**
  - The Incremental Model
  - The RAD Model
- **EVOLUTIONARY PROCESS MODELS**
  - Prototyping.
  - The Spiral model
  - The Concurrent Development Model
- **THE UNIFIED PROCESS .**
- **AGILE PROCESS/MODEL**

i. The Waterfall Model



The Waterfall Model

1) **Requirement Gathering and analysis**: Here requirements of the system or project that is to be developed are captured from client and all these requirements are documented in a requirement specification document to prepare a SRS. This SRS will be verified by the client and after acceptance next phase begins.

2) **System Design**: Here a design of the system is prepared. Plan system hardware and software requirements. It helps in defining the overall system architecture.

3)**Implementation**: Once the design of a system is prepared, the system is broken into small programs called units, which are integrated in the next phase. In this phase the software is coded and simple unit testing is performed.

The Waterfall Model

4)**Testing**: After testing all the units are integrated into a system and system testing is performed. This is used to find any bugs or failure in the system or to verify that system is built as per the requirements of client.

5)**Deployment**: Once the testing phase is done, the product or software is deployed or installed in the customer environment.

6)**Maintenance:** Maintenance is required to fix the issues and to release improved versions to enhance the product.

The Waterfall Model

Advantages:

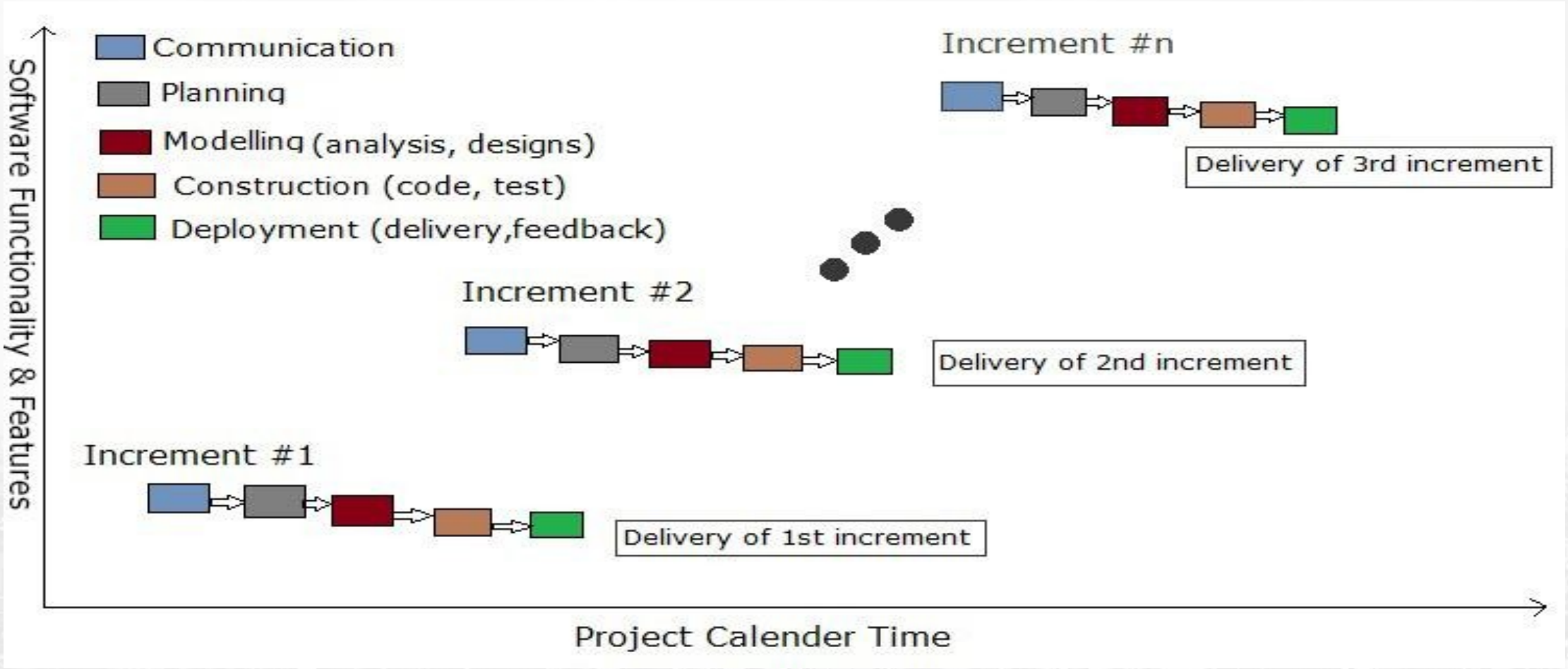
- Model is simple, easy to understand.
- Waterfall model is useful for smaller projects and it gives an appropriate result.
- System Requirements are documented and understood by projects team members.
- Each phase works independently and do not overlap the other phases.
- Customer interaction is only at the beginning and at the last phase of the project.

The Waterfall Model

Problems/Disadvantages:

- This model is not desirable for complex and bigger project where requirement changes frequently and risk factor is higher.
- This model also not good for ongoing projects.
- Customer interaction is less and it cannot adopt the changes in system requirements.
- This model requires more documentation which is not suitable for big projects.
- Customer feedback only at the end of the product.
- Small change makes lot of problems in the development

The Incremental Model



Incremental model

- The incremental model applies the waterfall model incrementally.
- The series of releases is referred to as "**increments**", each increment providing more functionality to the customers.
- The product is defined as finished only when it satisfies all of its requirements.
- It involves both development and maintenance.
- After **the first increment**, a **core product** is delivered, which can already be used by the customer. Based on customer feedback, a plan is developed for the next increments, and modifications are made accordingly.
- The incremental philosophy is also used in the **agile process model**.

Incremental model: Characteristics

Characteristics of Incremental Model

1. System is broken down into many **mini development projects**.
2. Partial systems are built to produce the final system.
3. First tackled **highest priority requirements.**
4. The requirement of a portion is frozen **once the incremented portion is developed**.

Tasks involved:

Communication: helps to understand the objective.

Planning: required as many people (software teams) work on the same project but different function at same time.

Modeling: involves business modeling, data modeling, and process modeling.

Construction: this involves the reuse software components and automatic code.

Deployment: integration of all the increments

Advantages & Disadvantages

Advantages of Incremental model:

i) Generates working software quickly and early during the software life cycle.

ii) This model is more flexible – less costly to change scope and requirements.

iii) It is easier to test and debug during a smaller iteration.

iv) In this model customer can respond to each built.

v) Lowers initial delivery cost.

Disadvantages of Incremental model:

i) Needs good planning and design.

ii) Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.

iii) Total cost is higher than other traditional model

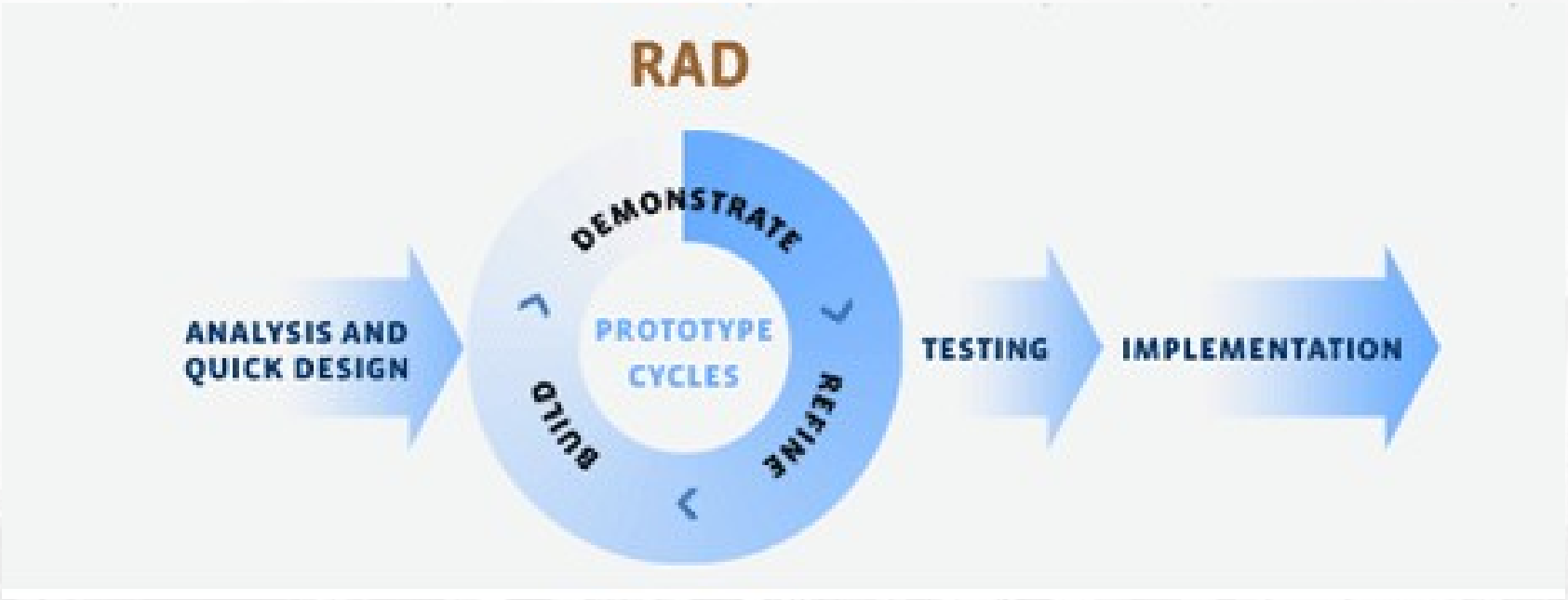
When to use Incremental Model ?

- When the requirements of the complete system are clearly defined and understood.
- Major requirements are defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available
- There are some high risk features and goals.

iii. RAD Model

- RAD is a **Rapid Application Development model**.
- Using the RAD model, software product is developed in a **short period** of time.
- The initial activity starts with the **communication** between customer and developer.
- **Planning** depends upon the initial requirements and then the requirements are divided into groups.
- Planning is more important to work together on different modules
- Rapid application development emphasizes working software and user feedback over strict planning and requirements recording.

RAD Model


RAD
ANALYSIS AND QUICK DESIGN
DEMONSTRATE
REFINE
BUILD
PROTOTYPE CYCLES
TESTING
IMPLEMENTATION

RAD Model

1. **Figure out the requirements : Identify why the** software or app is built and what the project is supposed to accomplish.

Involve users, developers, and designers to discuss the purpose of the system and a estimated project timeline. The budget is a strong constraint.

2. Build prototypes: Team will start working on building functional models right away. The engineers and designers will create and improve upon working prototypes

3. Get user feedback :

RAD calls for ongoing collaboration between your team and users in order to create a high-quality system. The users will be the ones providing feedback to improve your prototypes.

A general model of the design process

4. Do it again : Repeat steps two and three until you feel like your project is done or until you have all working parts assembled together to meet a client's requirements.

5. Test, test, test :

Run the system through different scenarios and make sure it accomplishes the system's goal.

RAD Model

Advantages of RAD Model

- For developing a large, complicated software, it helps to form more specialized teams.
- The process of application development and delivery are fast.
- This model is flexible, if any changes are required.
- Reviews are taken from the clients at the staring of the development hence there are lesser chances to miss the requirements.
- Always having something to show your client means you can get their feedback and quickly implement any changes

Disadvantages of RAD Model

- The feedback from the user is required at every development phase.
- This model is not a good choice for long term and large projects.

When is RAD useful?

- When a quick delivery of a product is needed for a customer.
- When there are going to be changes made to the prototype throughout the process before the final product is completed.
- When there are plenty of knowledgeable developers and engineers on hand and the customer must also remain committed to the process and the schedule.
- When either of these two components is not available, the RAD formula can fail.
- Eg. I*n the investment banking (IB) industry, most notably when applied to trading systems*

1.Communication: Developer and customer meet and discuss the overall objectives

2. Quick design

- Quick design is implemented when requirements are known.
- It includes only the important aspects like input and output format of the software and focusses on those aspects which are visible to the user rather than the detailed plan.
- It helps to construct a prototype.

3. Modeling quick design

- This phase gives the clear idea about the development of software
- It allows the developer to better understand the exact requirements.

4. Construction of prototype

- The prototype is evaluated by the customer itself.

5. Deployment, delivery, feedback

- If the user is not satisfied then it refines according to the requirements of the user.
- The process of refining the prototype is repeated until all the requirements of users are met.
- When the users are satisfied with the developed prototype then the system is developed on the basis of final prototype.

Prototyping Model

Advantages of Prototyping Model

- Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
- In the development process of this model **users are actively involved**.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.
- Gives quick user feedback for better solutions.
- It identifies the missing functionality easily and identifies confusing or difficult functions.

Disadvantages of Prototyping Model:

- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.
- It is a thrown away prototype when the users are confused with it.

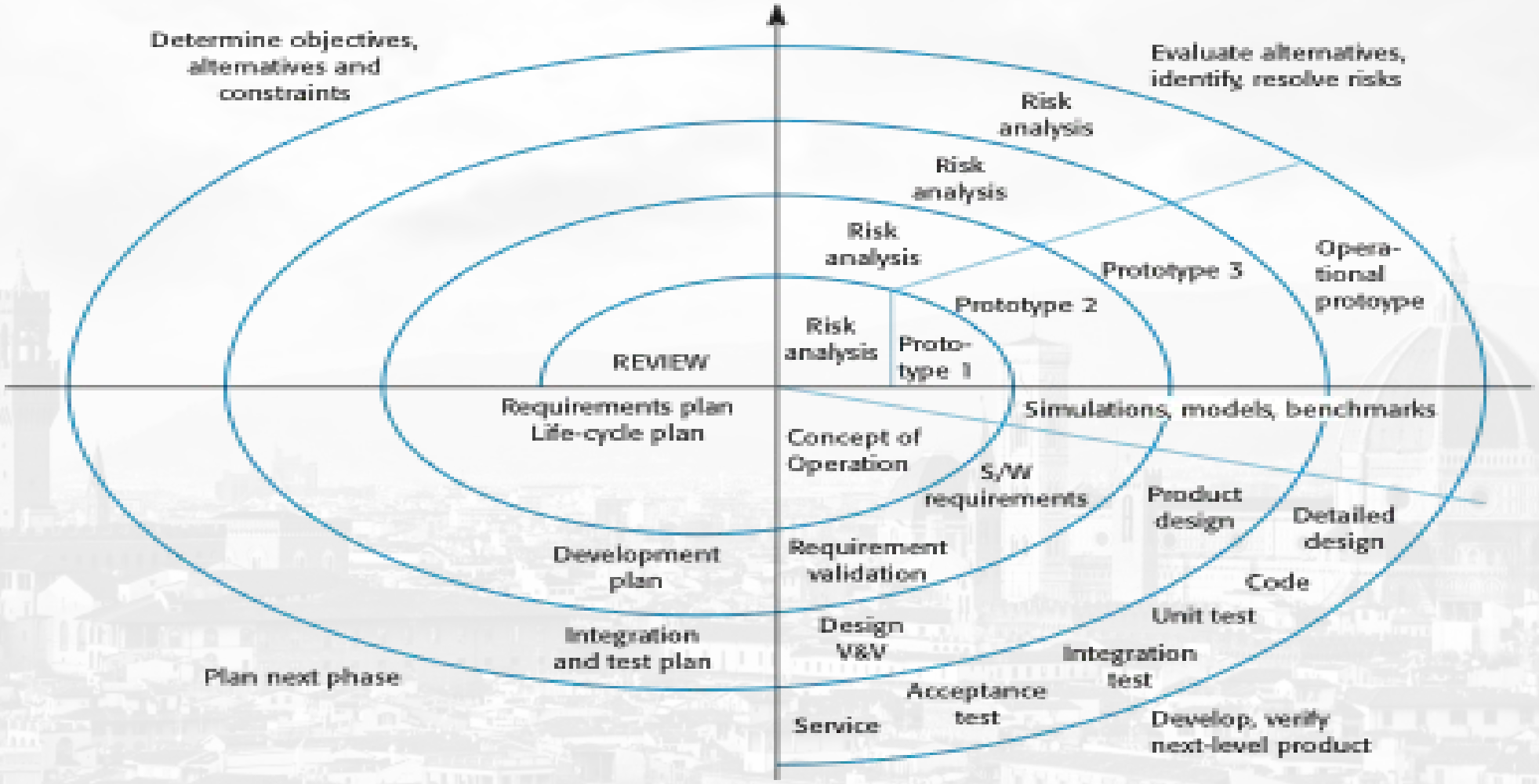
Prototype vrs Incremental Model

- Prototype Model: Instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements.
- Incremental model : the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a “multi-waterfall” cycle. Cycles are divided up into smaller, more easily managed modules.

v. Spiral Model

- Spiral model is a risk driven process model -which means that the overall success of a project highly depends on the risks analysis phase.
- Requires skills and expertise.
- It is used for generating the large software projects.
- In spiral model, an alternate solution is provided if the risk is found in the risk analysis, then **alternate solutions** are suggested and implemented.
- It is a combination of prototype and sequential model or waterfall model.
- In one iteration all activities are done, for large project's the output is small.
- Turns out costlier

Boehm's Spiral Model



Stages in Spiral Model

- Spiral Lifecycle Model was initiated by Boehm and is used while working with high risk projects.
- Features are combination of waterfall model and prototype model and activities executed in the form of a spiral.
- The entire project goes through the 4 stages now and then through each iteration known as a spiral.

Stage 1: Identification

- Starts with gathering the requirements , Identifications of system and sub-system with continuous communication.
- As soon as this spiral is accomplished, the project is deployed into the identified phase.

Stage 2: Design

- Basics of design in the baseline spiral which includes architectural, logical and physical design of the product.

Stage 3: Construct

- Build stage includes the construction of the existing product at each spiral.
- At that point in the resulting spirals, a model of the product with version number – “Build” is created. These builds are further sent to client for input for checking Proof of Concept

Stage 4: Assessment and Risk Analysis

- This model undergoes through the process of identifying, monitoring technical feasibility, estimation and risk management. Eg. such as schedule slippage and cost overrun.

Spiral model

- **Advantages of Spiral Model**
- It reduces high amount of risk.
- It is good for large and critical projects.
- It gives strong approval and documentation control.
- In spiral model, the software is produced early in the life cycle process.

Disadvantages of Spiral Model

- It can be costly to develop a software model.
- It is not used for small projects.

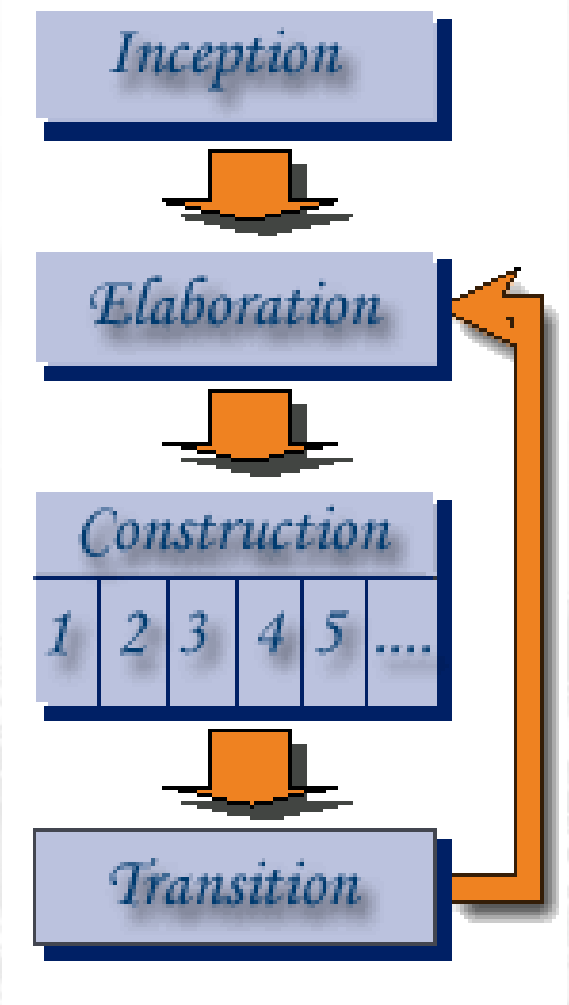
vi. The Unified Process

- A modern generic process derived from the work on the (Unified Modelling Language) UML and associated process.
- Brings together aspects of the 3 generic process models discussed previously.
- The UP recognizes that conventional process models present a single view of the process. In contrast, the UP is normally described from three perspectives:
  - A dynamic perspective, which shows the phases of the model over time.
  - A static perspective, which shows the process activities that are enacted.
  - A practice perspective, which suggests good practices to be used during the process.

Dynamic Perspective

- The UP is a phased model that identifies four discrete phases in the software process. The phases in UP are more closely related to business rather than technical concerns.
- Iteration within the UP is supported in two ways. Each phase may be enacted in an iterative way with the results developed incrementally.
- In addition, the whole set of phases may also be enacted incrementally, as shown by the looping arrow from Transition to Inception in the above diagram.

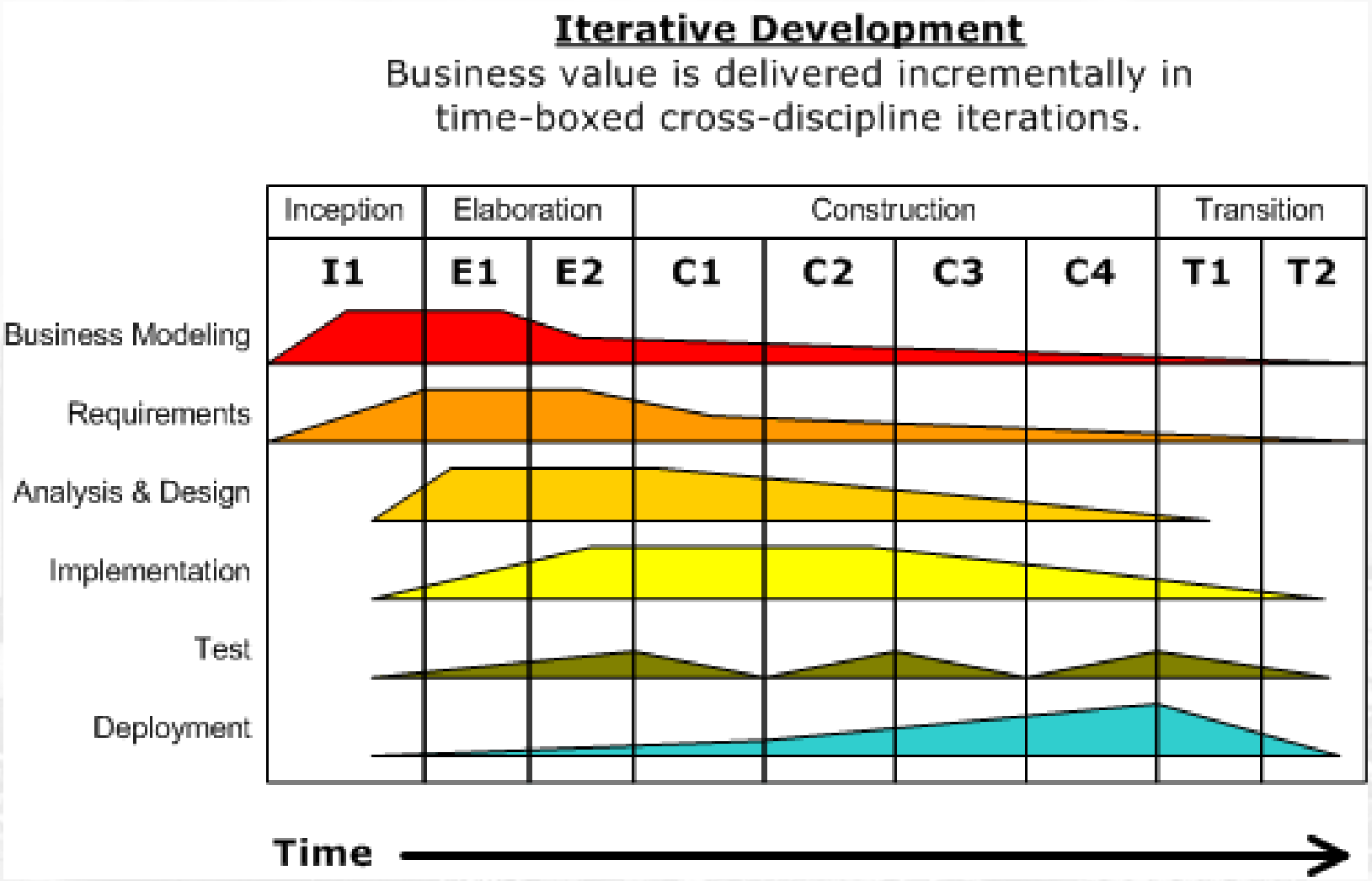
Phases in the Unified Process (Cont...)



Unified Process Work Products

UP PHASE	OBJECTIVE
Inception	Develop an approximate vision of the system, make the business case, define the scope, and produce rough estimates for cost and schedule.
Elaboration	Refine the vision, identify and describe all requirements, finalize the scope, design and implement the core architecture and functions, resolve high risks, and produce realistic estimates for cost and schedule.
Construction	Iteratively implement the remaining lower-risk, predictable, and easier elements and prepare for deployment.
Transition	Complete the beta test and deployment so users have a working system and are ready to benefit as expected.

Phases in the Unified Process (Cont...)



Static workflows in the Unified Process

This focuses on activities that take place during the development process – and called workflows

There are six core process workflows identified in the process and three core supporting workflows.

Workflow	Description
Business modelling	The business processes are modelled using business **use cases**.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.

Static workflows in the Unified Process

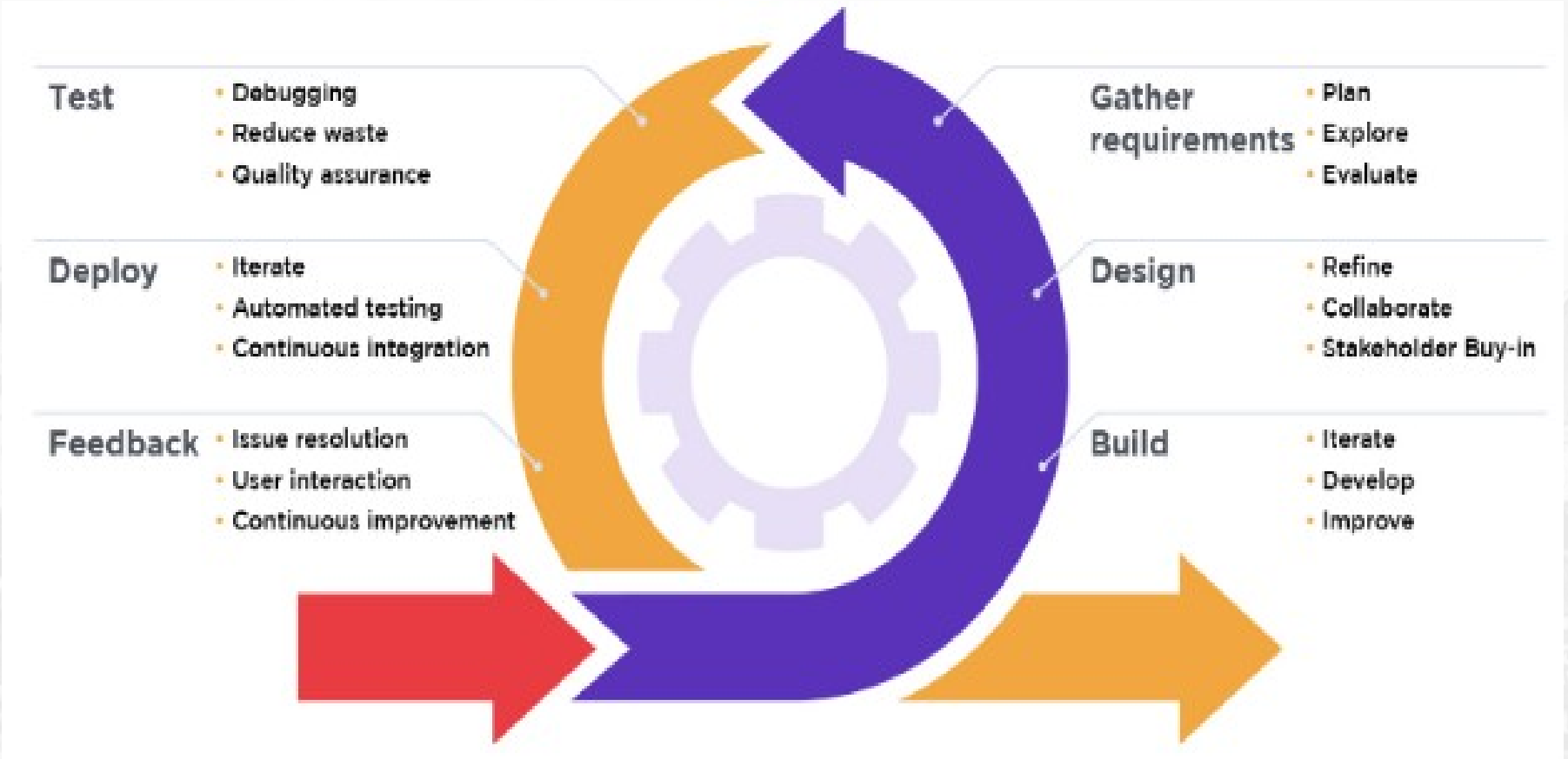
Workflow	Description
Testing	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system
Project management	This supporting workflow manages the system development
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

Unified Process good practice

- **Develop software iteratively :** Plan increments based on customer priorities and deliver highest priority increments first.
- **Manage requirements :**Explicitly document customer requirements and keep track of changes to these requirements.
- **Use component-based architectures :**Organize the system architecture as a set of reusable components.
- **Visually model software :** Use graphical UML models to present static and dynamic views of the software.
- **Verify software quality:** Ensure that the software meet's organizational quality standards.
- **Control changes to software :**Manage software changes using a change management system and configuration management tools.

vii. The Agile Model

- Agile Model



Agility

- Effective (rapid and adaptive) response to change (team members, new technology, requirements)
- Effective communication in structure and attitudes among all team members, technological and business people, software engineers and managers
- Drawing the customer into the team. Eliminate “us and them” attitude. Planning in an uncertain world has its limits and plan must be flexible.
- Organizing a team so that it is in control of the work performed
- Eliminate all but the most essential work products and keep them lean.
- Emphasize an incremental delivery strategy as opposed to intermediate products that gets working software to the customer as rapidly as feasible.

Agile Process/Model

- Is driven by customer descriptions of what is required (scenarios).
- Some assumptions:
  - Recognizes that plans are short-lived(some requirements will persist, some will change, Customer priorities will change)
  - Develops software iteratively with a heavy emphasis on construction activities (design and construction are interleaved, hard to say how much design is necessary before construction. Design models are proven as they are created.)
  - Analysis, design, construction and testing are not predictable.
  - Thus has to Adapt as changes occur due to unpredictability
  - Delivers multiple ‘software increments’, deliver an operational prototype or portion of an OS to collect customer feedback for adaption.

Agile Process/Model

1. Our highest priority is to satisfy the customer requirements through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.

Agile Process/Model

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

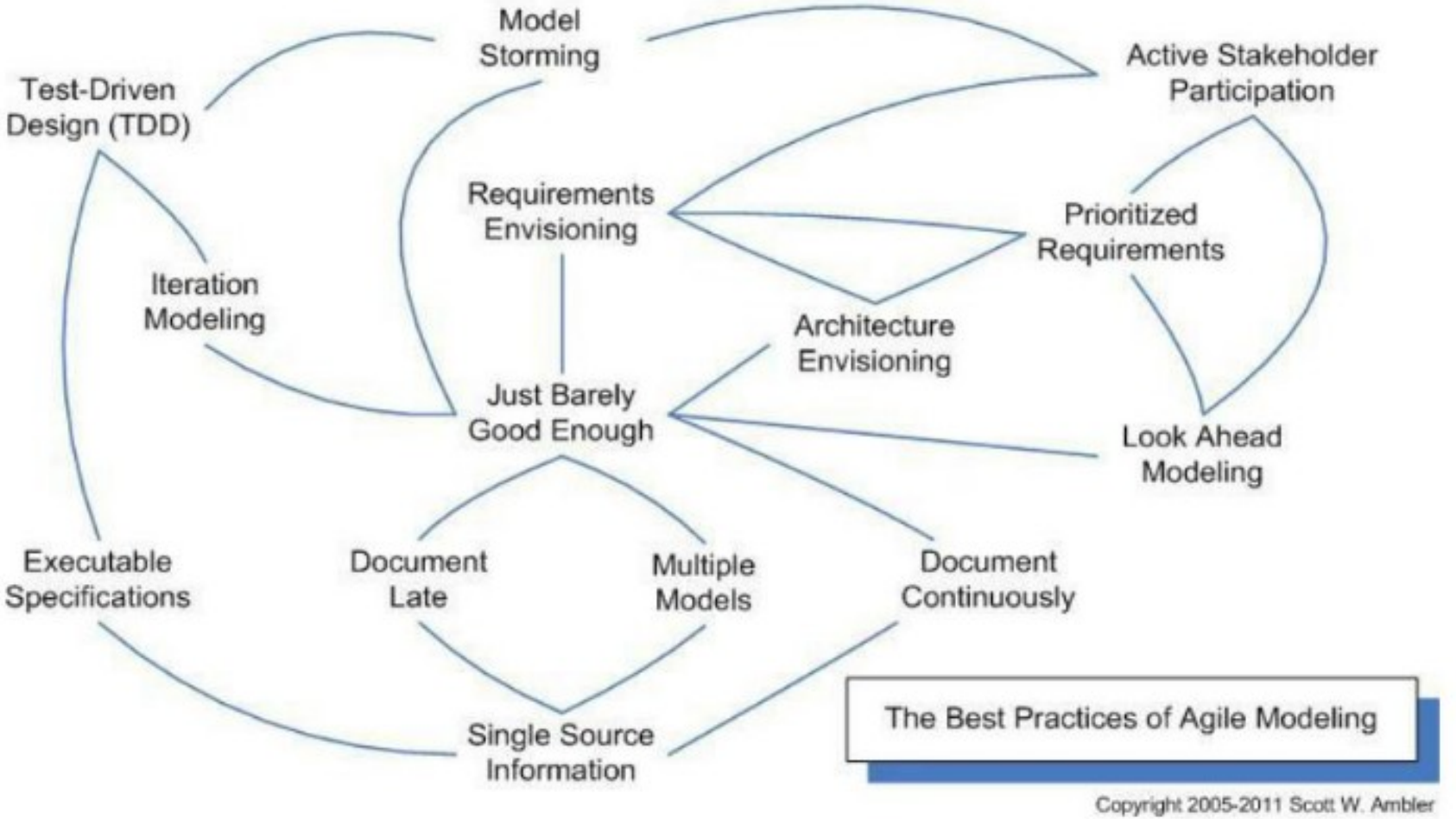
9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity – the art of maximizing the amount of work not done – is essential.

11. The best architectures, requirements, and designs emerge from self–organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Modeling



https://image1.slideserve.com/1584785/slide27-l.jpg

Other Process Models

- Component based development—the process to apply when reuse is a development objective (like spiral model)
- Formal methods—emphasizes the mathematical specification of requirements (easy to discover and eliminate ambiguity, incompleteness and inconsistency)
- Aspect Oriented software development (AOSD)—provides a process and methodological approach for defining, specifying, designing, and constructing aspects

How to choose an appropriate process model?

- **Characteristics of the software to be developed:** . for product and embedded development, the Iterative Waterfall model can be preferred. The evolutionary model is suitable to develop an object-oriented project. User interface part of the project is mainly developed through prototyping model.
- **Characteristics of the development team:** If the development team is experienced in developing similar software, then even an embedded software can be developed using the Iterative Waterfall model. If the development team is entirely novice, then even a simple application may require a prototyping model.
- **Risk associated with the project:** If the risks are few and can be anticipated at the start of the project, then prototyping model is useful. If the risks are difficult to determine at the beginning of the project but are likely to increase as the development proceeds, then the spiral model is the best model to use.
- **Characteristics of the customer:** If the customer is not quite familiar with computers, then the requirements are likely to change frequently as it would be difficult to form complete, consistent and unambiguous requirements. Thus, a prototyping model may be necessary to reduce later change requests from the customers.
  - the evolutionary model is useful as the customer can experience a partially working software much and reduces the customer's trauma of getting used to an entirely new system.

Comparison

Parameter / Process Model→	Waterfall Model	Incremental Model	Prototype Model	Rad Model	Spiral Model	Agile Model	Xp programming
Clear Requirement Specifications	Initial level	Initial level	At medium level	Initial level	Initial level	Change incrementally	Initial level
Feedback from user	No	No	Yes	No	No	No	Yes
Speed to change	Low	High	Medium	No	High	High	High
Predictability	Low	Low	High	Low	Medium	High	High
Risk identification	At initial level	No	No	No	Yes	Yes	Yes
Practically implementation	No	Low	Medium	No	Medium	High	High
Loom	Systematic sequence	Iterative sequence	Priority on customer feedback	Use readymade component	Identification of risk at each stage	Highly customer satisfaction and incremental development[09]	Customer satisfaction and incremental development
Any variation done	Yes-v model	No	No	No	Yes-win win spiral[6]	No	No
Understandability	Simple	Intermediate	Intermediate	Intermediate	Hard	Much complex	Intermediate
Precondition	Requirement clearly defined	Core product should clearly define	Clear idea of Quick Design	Clean idea of Reuse component	No	No	No
Usability	Basic	Medium	High	Medium	Medium	Most use now a days	medium
Customer priority	Nil	Nil	Intermediate	Nil	Intermediate	High	Intermediate
Industry approach	Basic	Basic	Medium	Medium	Medium	High	Medium
Cost	Low	Low	High	very high	Expensive	Much Expensive	High
Resource organization	Yes	Yes	Yes	Yes	No	No	Yes
Elasticity	No	No	Yes	Yes	No	Very high	Medium

How to choose an appropriate Process Model ?

- Some key points to consider :
- Observing the domain problem at hand
- The amount of resources available to solve the problem (time, intellectual capital and money could help)
- The culture of the organization being served
- Assess the needs of Stakeholders

Choosing an appropriate Process Model

Factors	Waterfall	V-Shaped	Evolutionary Prototyping	Spiral	Iterative and Incremental	Agile
Unclear User Requirement	Poor	Poor	Good	Excellent	Good	Excellent
Unfamiliar Technology	Poor	Poor	Excellent	Excellent	Good	Poor
Complex System	Good	Good	Excellent	Excellent	Good	Poor
Reliable system	Good	Good	Poor	Excellent	Good	Good
Short Time Schedule	Poor	Poor	Good	Poor	Excellent	Excellent
Strong Project Management	Excellent	Excellent	Excellent	Excellent	Excellent	Excellent
Cost limitation	Poor	Poor	Poor	Poor	Excellent	Excellent
Visibility of Stakeholders	Good	Good	Excellent	Excellent	Good	Excellent
Skills limitation	Good	Good	Poor	Poor	Good	Poor
Documentation	Excellent	Excellent	Good	Good	Excellent	Poor
Component reusability	Excellent	Excellent	Poor	Poor	Excellent	Poor

Summary

- Process is a means to achieve project objectives of high Quality
- Software process models are abstract representations of these processes
- Process models define generic process, which can form basis of project process
- Process typically has stages, each stage focusing on an identifiable task
- Many models for development process have been proposed
- Waterfall model, Evolutionary development and component-based software Engineering, Iterative process models are some process model.
- The Rational Unified Process is a generic process model that separates activities from phases
- A prototype can be used to give end-users a concrete impression of the system's capabilities

MIT-WPU

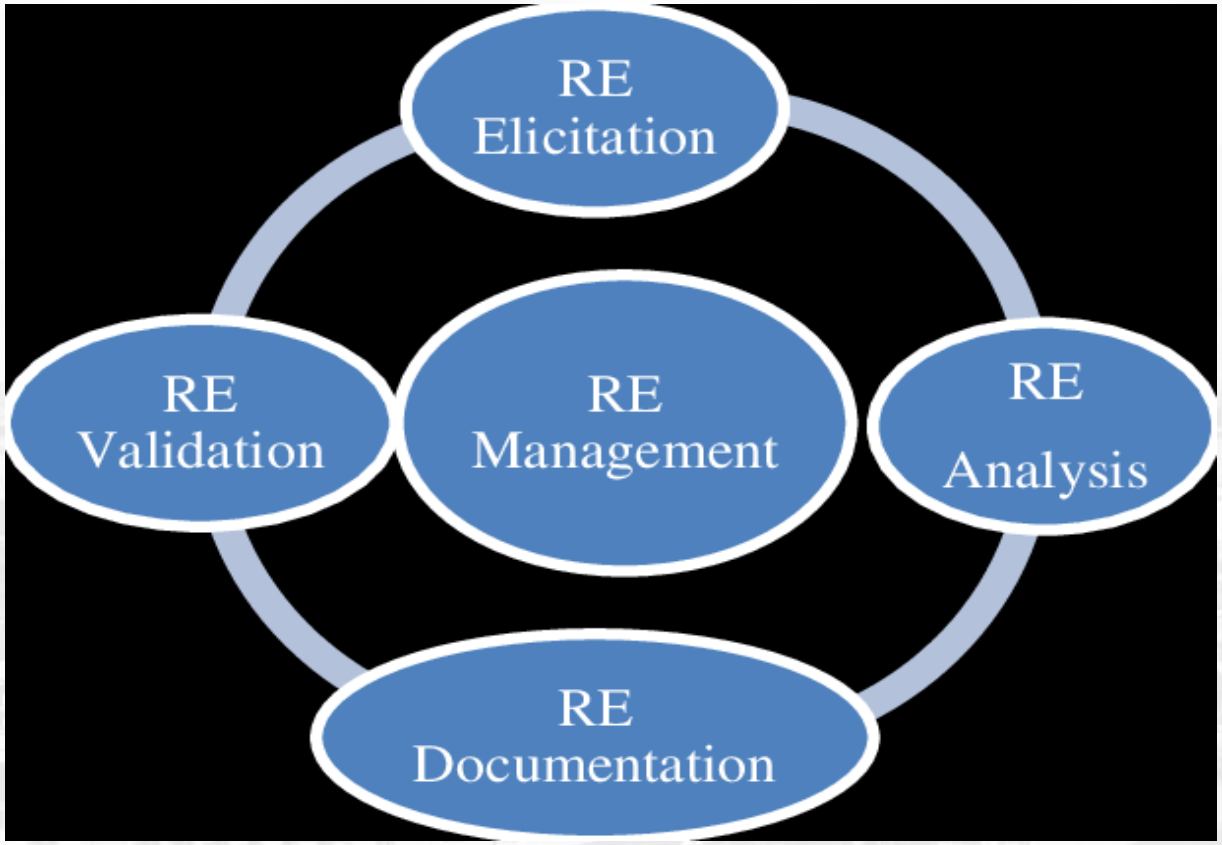
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Requirement Engineering

Software Development Life Cycle

The Requirement Engineering

Requirements engineering is a process of gathering and defining of what the services should be provided by the system.



What is Requirements Engineering?

- Requirements are description of features and functionalities of the system and convey the expectations of users from the software product.
- Requirements engineering refers to the process of defining, documenting and maintaining requirements in the engineering design process.
- It is a common role in systems engineering and software engineering.
- The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirement Definitions and Specifications

Example

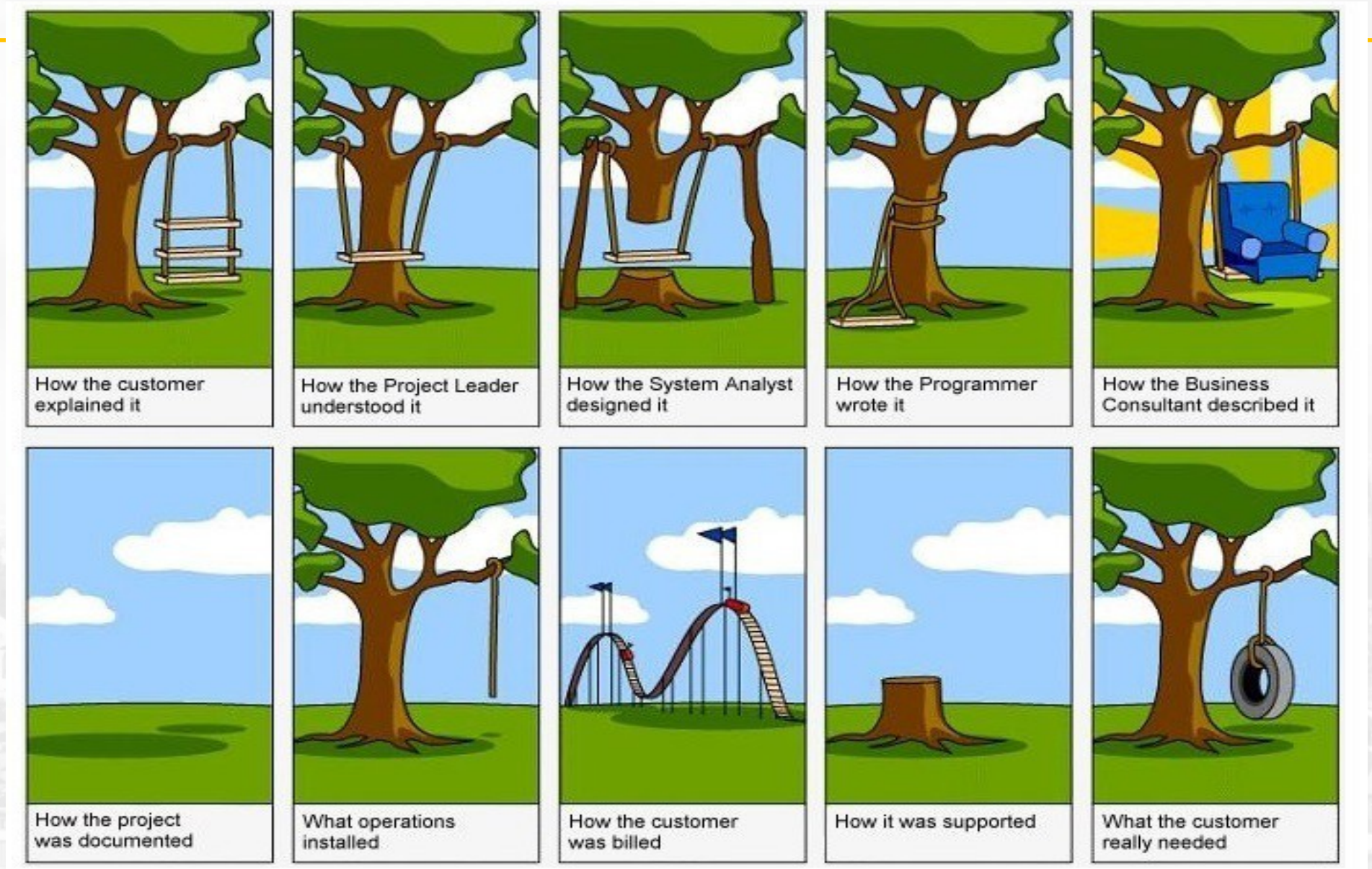
Requirements definition

1. The software must provide a means of representing and accessing external files created by other tools.

Requirements specification

1.1 The user should be provided with facilities to define the type of external files.
1.2 Each external file type may have an associated tool which may be applied to the file
1.3 Each external file type may be represented as a specific icon on the user's display.
1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.
1.5 When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

Ugly Face of Requirement Engineering



Requirement Engineering Steps

- RE has overlapping steps :
  - **Inception** - in which the nature and scope of the system is defined.
  - **Elicitation** - in which the requirements for the software are initially gathered.
  - **Elaboration -** in which the gathered requirements are refined.
  - **Negotiation** -in which the priorities of each requirement is determined, the essential requirements are noted, and, conflicts between requirements are resolved.
  - **Specification** - in which the requirements are gathered into a single product, being the result of the requirements engineering.
  - **Validation -** in which the quality of the requirements (i.e., are they unambiguous, consistent, complete, etc.), and the developer's interpretation of them, are assessed.
  - **Management -** in which the changes that the requirements must undergo during the project's lifetime are managed.

Requirements Engineering Steps

- Inception— Ask a set of questions that establish …
  - Basic understanding of the problem
  - The nature of the solution that is desired
  - The effectiveness of preliminary communication and collaboration between the customer and the developer
  - The people who want a solution -Identify Stakeholders : Stakeholders can affect or be affected by the application's actions, objectives and policies. End Users, Build Team and Authorities
  - Recognize multiple points of view

Requirements Engineering- Steps

- Elicitation
  - Elicit requirements , identify problems from all stakeholders
  - Propose elements of the solution
  - The scope and negotiate different approaches,,
  - Understanding of the problem and volatility (requirements change over time)
  - Specify a preliminary set of solution requirements

Requirements Engineering- Steps

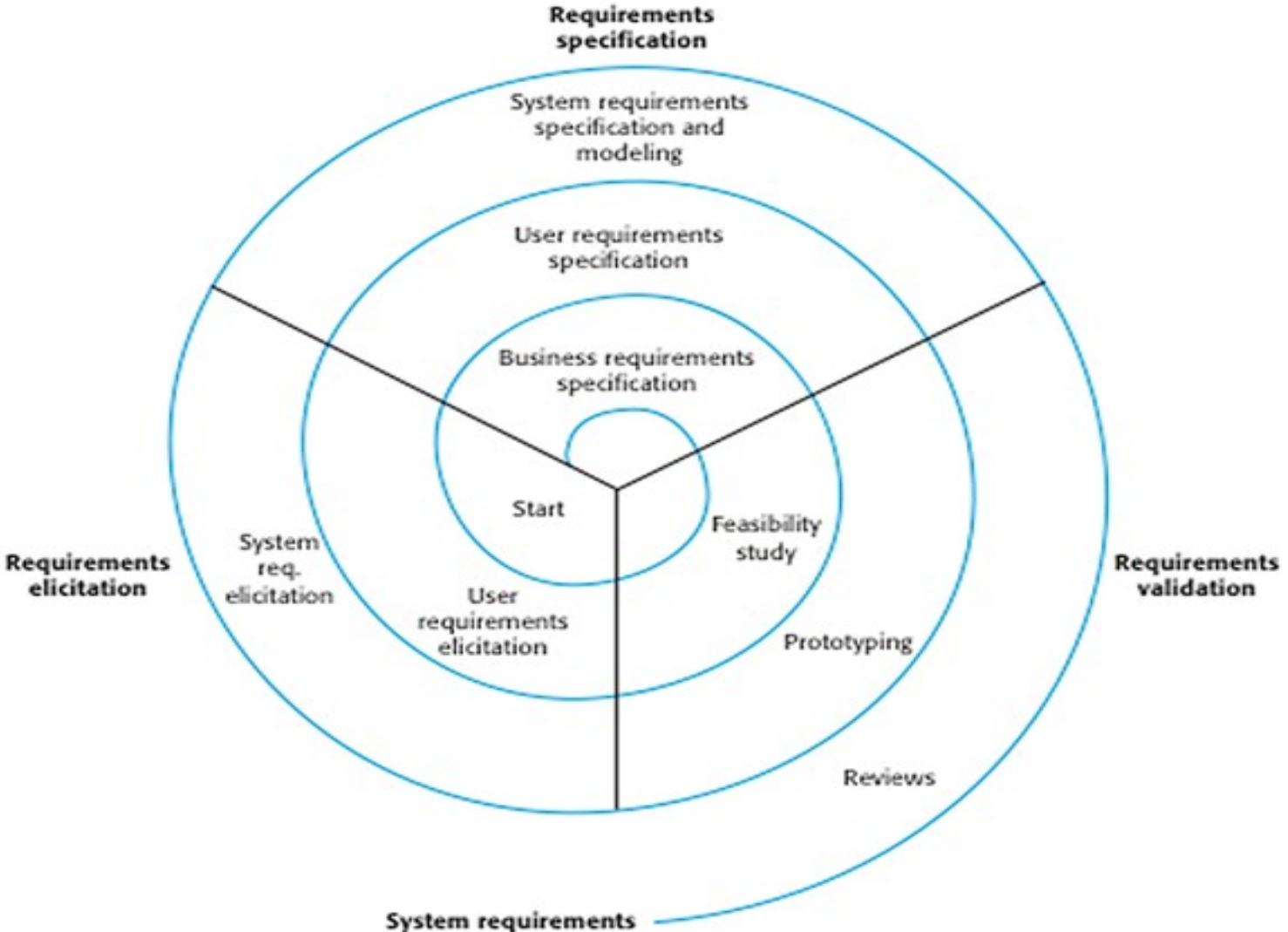
- Elaboration
  - Create an analysis model that identifies data, function and behavioral requirements.
  - It is driven by the creation and refinement of user scenarios that describe how the end-user will act with the system.

Requirements Engineering- Steps

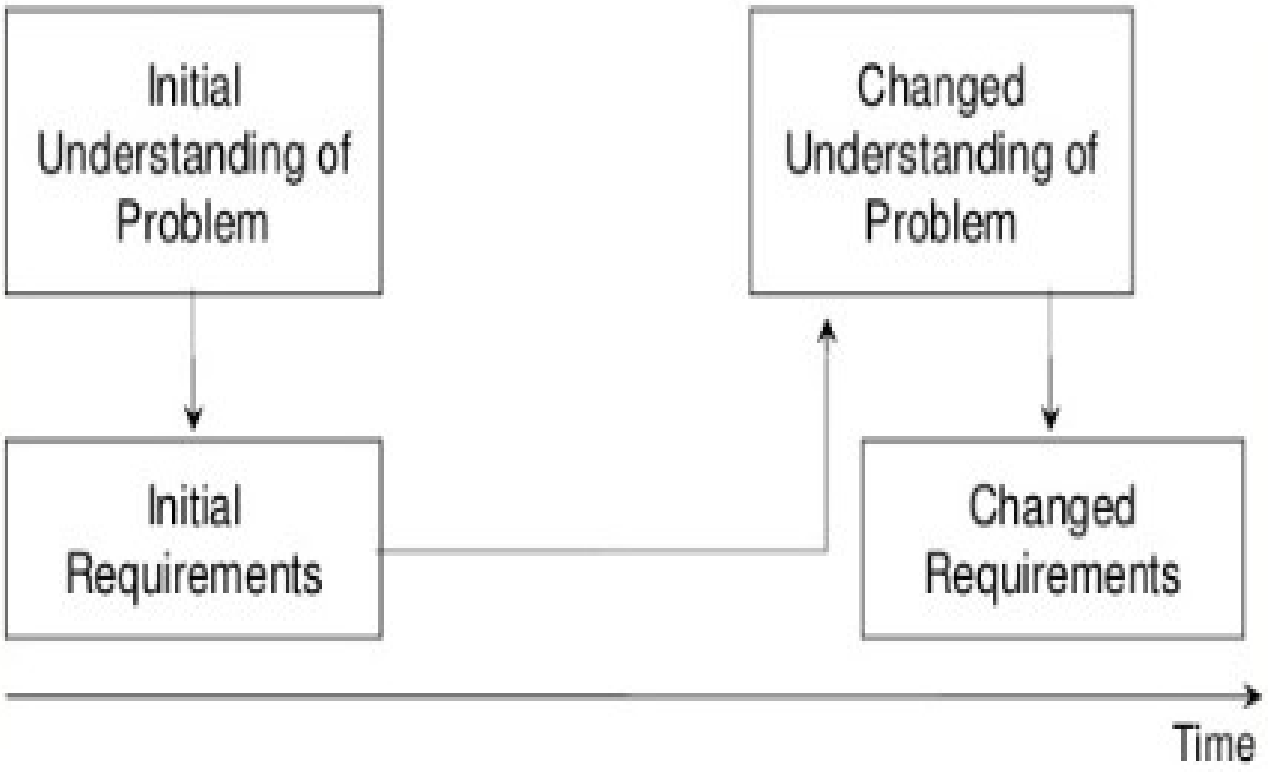
- Negotiation
  - Identify Conflicting Requirements and reconcile these conflicts through a process of *negotiation*.
  - Stakeholders are asked to rank requirements and then discuss conflicts in priority.
  - Risks in each requirement are identified and analyzed.
  - Agree on a deliverable system that is realistic for developers and customers.
- Specification : SRS or a prototype
  - It is the final work produced by the RE. It is serves as the foundation for subsequent S.E. activities.

- Validation
  - A review mechanism that looks for
    - Errors in content or interpretation
    - Areas where clarification may be required
    - Missing information
    - Inconsistencies (a major problem when large products or systems are engineered)
    - Conflicting or unrealistic (unachievable) requirements.
- Management
  - Set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds.

The process of requirements engineering

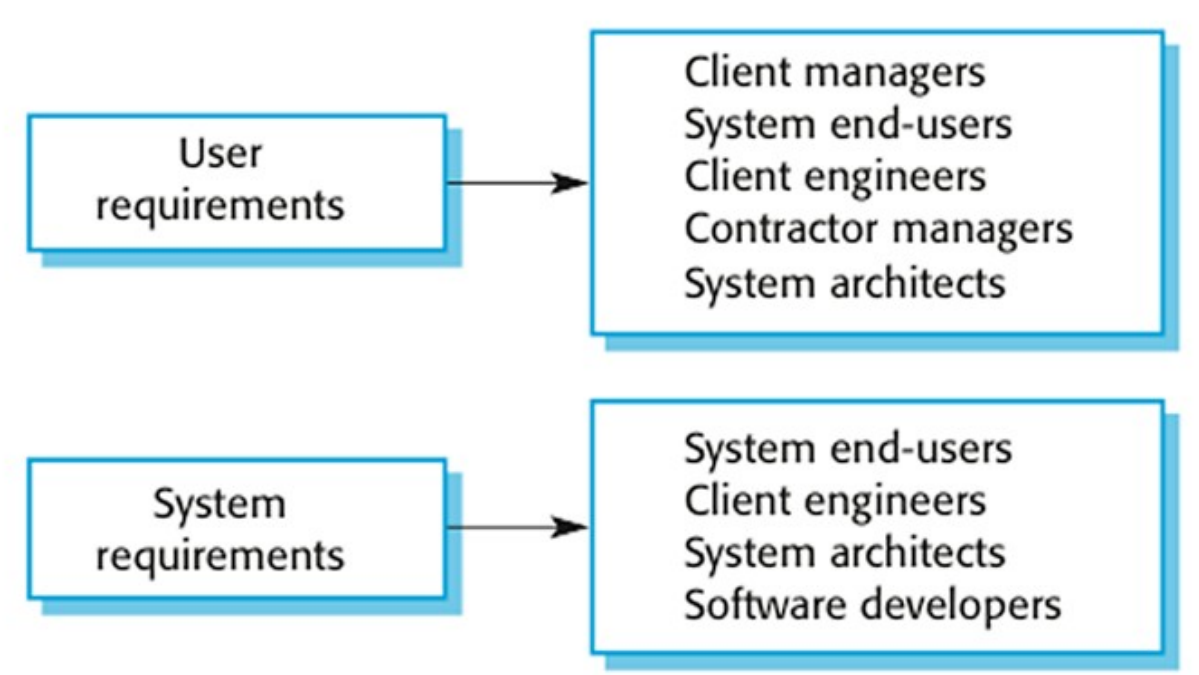


Requirements Evolution



User and System Requirements

- User need a high-level statements of the requirements, while system developers need a more detailed system specification.
- Having different level of details is useful because it communicates information about the system being developed for different types of readers.



User and System Requirements

- User Requirements : describes the services that the system should provide and the constrains under which it must operate.
  - more of generic requirements and Readable by everybody
  - Services and constraints of the system
  - In natural language or diagrams
  - Serve business objectives
- System Requirements : gives a more detailed description of the system services and the operational constrains (how system will be used, programming languages etc)
  - This level of detail is needed by those who are involved in the system development
  - Useful for the design and development
  - Precise and cover all cases
  - Structured presentation

Example

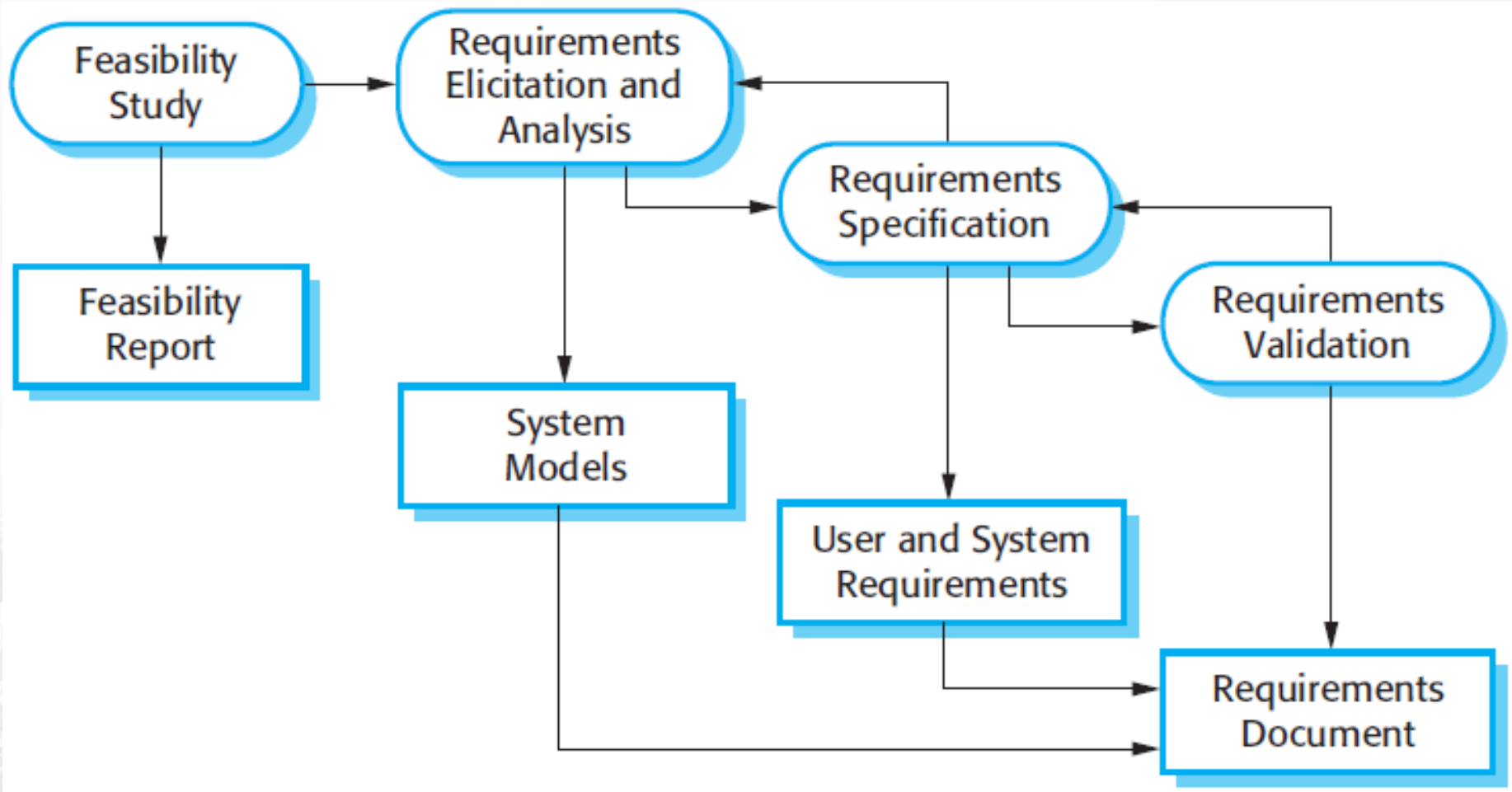
- **User requirement**: The library system should provide a way to allow a patron to borrow a book from the library.
- **System requirement**: The library system should provide a withdraw interaction that allows a patron to withdraw a book given the isbn and copy number of the book to be withdrawn. The interaction fails if: the book is already withdrawn, the book is not in the library's collection, the patron has already withdrawn 5 books, the book is on hold by someone else.
- **Software Specification**: A detailed software description which can serve as a basis for a design or implementation. Written for developers

Business requirements	User requirements	Software requirements
Outline measurable goals for the business.	Reflect specific user needs or expectations.	Identify features, functions, non-functional requirements and use cases.
Define the why behind a software project.	Describe the who of a software project.	Delve into the how of a software project.
Match project goals to stakeholder goals.	Highlight how users interact with it.	Describe software as functional modules and non-functional attributes.
Maintain a BRD with requirements, updates or changes.	Create a URS, or make them part of the BRD.	Compose an SRS, and, optionally, an FRS.

RE Process

- Requirement Engineering is the process of defining, documenting and maintaining the requirements.
- A process of gathering and defining service provided by the system.
  - It ensures your software will meet the user expectations, and ending up with a high quality software.
  - It's a critical stage of the software process as errors at this stage will reflect later on the next stages, which definitely will cause you a higher costs.
- At the end of this stage an SRS is produced and validated with the stakeholders.

The RE process



RE Process

- **Feasibility study:**
  - Check is made if the identified can be achieved using the given conditions
  - This step should be cheap and quick;
- **Requirements elicitation and analysis:**
  - Deriving the system requirements through observation of existing systems, discussions with stakeholders, etc.
  - Can involve development of a system models and prototypes to understand the specified system
- **Requirements specification:**
  - It's the activity of writing down the information gathered during the elicitation and analysis activity into a document that defines a set of requirements.
  - Two types of requirements may be included in this document; user and system requirements.
- **Requirements validation:**
  - It's the process of checking the requirements for realism, consistency and completeness.
  - Goal is to discover errors in the requirements document.
  - When errors are found, it must be modified to correct these problems.

Types of Requirements

- **Functional requirements**
  - Services the system should provide
  - What the system should do or not in reaction to particular situations
- **Non-functional requirements**
  - Constraints on the services or functions offered by the system
  - Examples: Timing constraints, constraints on the development process (CASE, language, development method…), standards etc
- **Domain requirements**
  - From the application domain of the system
  - May be functional or non-functional
  - Examples: Medicine, library, physics, chemistry

Bad Requirements

- **Missing Requirements** –A functionality that is totally missing from the documentation.
  - "Error messaging" in case of data validation
  - not detailing the need for a certain link available as per the access rules on an application.
- **Conflicting Requirements** – When two or more requirements expect the system to do different things that can't possibly be done at the same time.
  - the business stakeholder might want to retrieve a 1000 records at a time in real-time whereas the technology stakeholder knows this is practically impossible and not feasible.
- **Incomplete/Unclear Requirements** – Requirements that lack all the necessary information constitute this lot.
  - "The system should have the capability to filter search results". The details around filter criteria is not been provided and thus raises unnecessary queries.
- **Ambiguous Requirements** – A requirement statement that can be interpreted in different ways by different people.

Ambiguous Requirements

- *A client needed to be able to upload "large files." A solution was found where the platform could handle the "large files." Client was assured a solution was found and testing began. Client gave a feedback that the solution was not meeting their needs, was freezing and full of bugs.*
  - *To the client "large" meant 20-50GB and to our team and the platform provider "large" meant up to 5GB.*
- *Trigger an automatic log out when a user attempts to download up to 5 times. Display is up to large screen size.*
  - Is this 5 times including the 5th time or 4 times?
  - the page has a document that is in 5 parts
  - For the screen size, 720px may be large for some and 1280 may be large for some.

Requirements Checking

- Validity. Does the system provide the functions which best support the customer's needs?
- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

Requirements: Functional

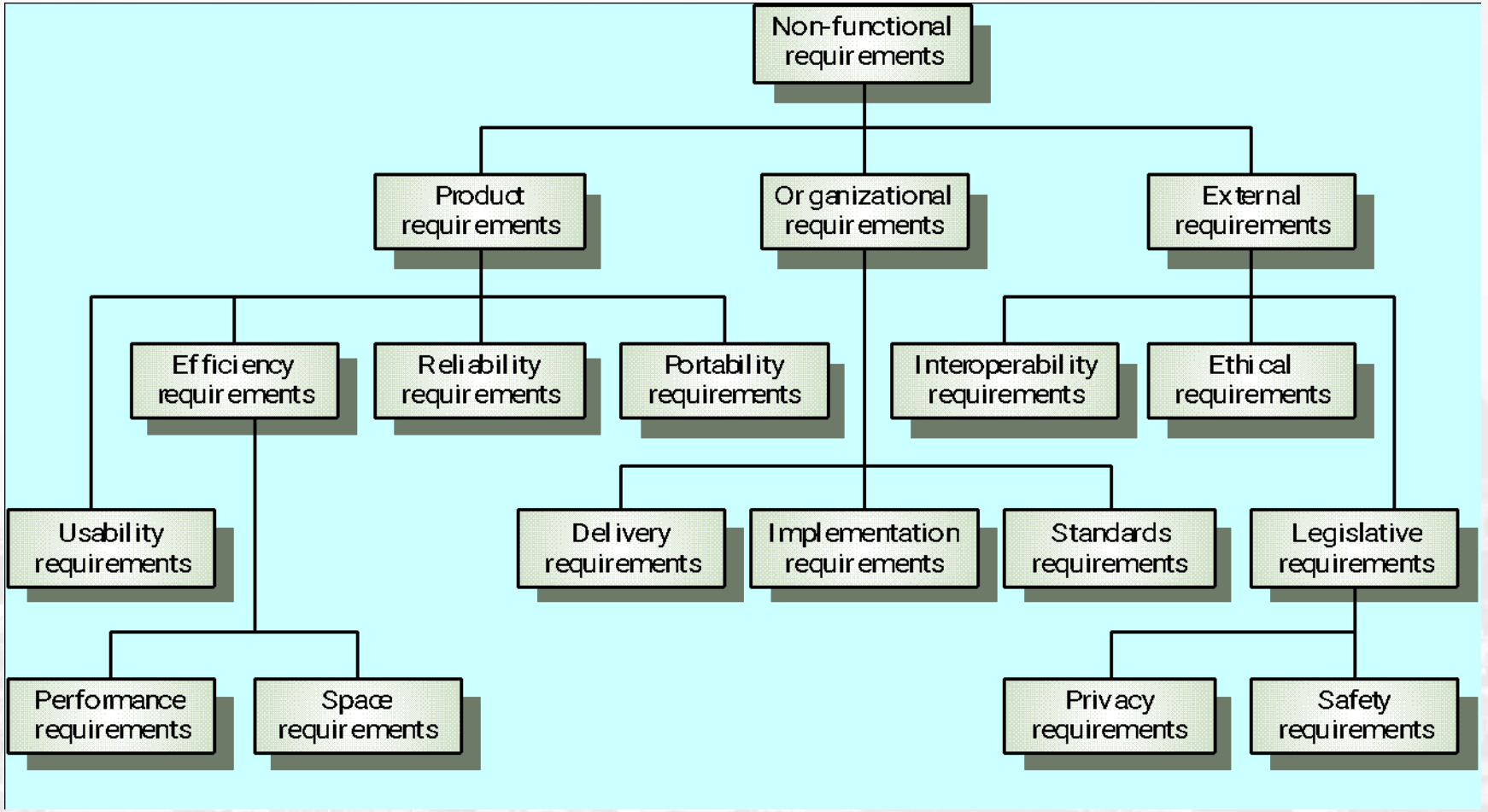
- Functional requirements:
  - Depend on the system, the software, and the users
  - Can be expressed at different levels of detail (user/system requirements)
  - For a system, it is desirable to have a complete and consistent set of functional requirements
    - *Completeness*: all required system facilities are defined
    - *Consistency*: there are no contradictions in requirements

Requirements: Non-functional

- Non-functional requirements:
  - Many apply to the system as a whole
  - More critical than individual functional requirements
  - More difficult to verify
- Kinds of non-functional requirements:
  - Product requirements
  - Organizational requirements
  - External requirements

Requirements: Non-functional

- A classification of non-functional requirements:



Requirements: Non-functional

- Metrics that can be used to quantitatively specify and verify non-functional requirements

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

References

- Roger S Pressman, Software Engineering: A Practitioner's Approach, Mcgraw-Hill, ISBN: 0073375977, Seventh Edition, 2014
- Ian Sommerville, — Software Engineering‖, Addison and Wesley. 9th Ed., 2011.
- Pankaj Jalote, Software Engineering: A Precise Approach, Wiley India.2010.

Disclaimer:

a. Information included in this slides came from multiple sources. We have tried our best to cite the sources. Please refer to the References to learn about the sources, when applicable.

b. The slides should be used only for academic purposes (e.g., in teaching a class), and should not be used for commercial purposes.