PRELIM

1.1. Introduction to Software Engineering

Software has revolutionized how people live everyday. Software is a product of Software Engineering which both are further explained in the succeeding sections. It also discusses what makes a good software. As future software engineers, you are still bound to certain degree of ethics to guide you in creating a good software.

Software

Is a set of instructions or executable codes, libraries, and documentations. It can be developed for specific customer or for general market. There are two types of software

System Software

1. Operating systems such as Windows for Microsoft, iOS for Apple, Linux, Android
2. Firmware
3. Drivers

Application Software

1. Browsers such as Google Chrome and Edge
2. Word Processor such as Microsoft Word
3. Spreadsheet such as Microsoft Excel
4. Databases such as MySQL and Oracle

It is said that a good software should deliver the required functionality and performance to the user and should be **maintainable**, **dependable** and **secured, efficient,** and **acceptable**.

Attributes of a Good Software

- 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 and 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 utilization, 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.

That is why for us to build a good software, we must understand and apply the different concepts and techniques available in Software Engineering.

Software Engineering

Software engineering is an engineering discipline that is concerned with all aspects of software production from early stages of system specification through maintaining the system after it has gone into use.

When you are already working out a software the following processes will always be encountered regardless of the model that you are going to follow

1. **Planning**
2. **Analysis and design**
3. **Implementation** by writing your codes
4. **Validation/Testing** by employing different types of testing
5. **Deployment** where you ask your users to use the software
6. **Maintenance**

Going back to the definition of the software (take note this is the byproduct of software engineering), there are two types of software that can be developed

Generic Products

- Usually these are standalone applications such as word processors, databases, and system with generic purpose such as library system and accounting system.
- I would like to emphasize the SaaS (or the software as a service) that is available when cloud computing becomes a popular some years ago. The concept is you pay for what you use. Some arrangement also comes in a form of membership package such as FREE, Silver, or Gold members. Of course, the gold members will be able to utilize all features available that the silver package. And the free will have limited features available.

Customized Products

- Usually this is where you come in as software engineer
- These are software that are specific to customers because they have unique needs than the generic products.
- Examples: system to support a business process such as traffic control of course this is not generic because they have different configurations by places.

Software Engineering vs Computer Science vs System Engineering

- **Computer science** – focuses on theories

and methods that underlie computer and software systems. Theories such as data representation, automation, efficiency in coding and processes, as well algorithms.

- **Software engineering** – concerned with practical problems of producing software. Some knowledge of CS is essential for Software Engineers especially when dealing already with non-functional requirements.
- **Systems engineering** – concerned with all aspects of computer based systems development including hardware, software, and process engineering. So, software engineering is simply part of systems engineering.

Issues Affecting Different Types of Software

	Heterogeneity	Business and Social Change	Security and Trust
Definition	ability of the software to operate on diversity of environment.	business and society are changing fast and new technologies becomes available.	how to assure that malicious users cannot attack software and information securit is maintained.
Example	Deploying application to different mobile operating system	Government policies affecting business processes	How different organizations cope up with new threat of hacking

Software Engineering Diversity

Software Engineering is very diverse that the following can be one of the software that you will be working on in the future

1. Standalone application
2. Interactive transaction-based applications
3. Embedded control systems
4. Batch processing systems
5. Entertainment systems
6. Systems for modeling and simulation
7. Data collection systems
8. Systems of systems – this is interesting because you will be studying how to make different software and hardware work together. The most challenging part are integrating legacy systems with new ones

Software Engineering Ethics

1. **Confidentiality**

Respect confidentiality of clients or employers regardless of availability of confidentiality agreement.

In the industry, we call this as Non-Disclosure Agreement. This is an agreement between client-vendor (third party provider) for non-disclosing any details to non-

stakeholders.

2 Competence

One should not knowingly accept work that is outside of competence

One should be honest that if it is not their forte or specialization, then they should not take work because it is too risky. If you notice software providers always attached to their name, the level of competence they were able to achieve such as they are Oracle partner therefore, they are in some sort of high confidence in Oracle products.

3 Intellectual property rights

Observance of local laws and international laws on the use of Intellectual Property such as patents and copyright.

Licenses must also be respected if available.

4 Computer misuse

One should not use technical skills to misuse other people's computer. In the offices, no game playing if there is a policy on games as well as observing strict compliance of social media, visiting unsecured websites due to possibility downloading viruses and malwares.

1.2. Software Processes 1 of 2

Software Engineering Fundamental Activities

You will notice that the four activities will be common to

the different software process models

1. **Software specification**
   - Software specification starts with Feasibility Study to initially check if the project can materialize in terms of implementation, cost, objectives of the organization, and most especially it will provide benefit to the organization.
   - Following feasibility study is the requirements elicitation and analysis, and requirements specification
   - Last will be requirements validation
2. **Software design and implementation**
   - This activity focuses on executing the system specification into an executable system or your development stage. Before the development, you will also be doing the
     - Architectural design to present the structure of the system
     - Interface design to see how each component of the system interacts
     - Component design to see how each system components operate
     - Database design to see how data are stored and be related to database
   - Note that at this stage, your client may not be interested in this technicality, but as software engineers and software developers, this is crucial because these designs are a lot closer to how you perform your development.

3 **Software validation**

- Software validation is simply testing the output of the previous stage. However, it is not purely testing. You must learn how to plan how you conduct your testing. In planning, you consider what will be used, who will conduct the testing, what will be the acceptable criteria (although this should be defined during the software specification stage), when, where to conduct testing and other details that are important to planning.
- Here you will also apply the different types of testing such as component-based or system testing.
- The final process here will be acceptance testing by using actual user data to validate if it is the same output/experience-based on the test data.

4 **Software evolution**

- This refers to the maintenance of the system. As explained before, one of the attributes of good software is maintainability.
- In incremental models and agile development, this is very much in use because you are already planning which among the remaining features will be on the next iteration.

System design and software design

During the design phase, you will often encounter system design and software design. There are few differences between the two even if they appear to be the same.

- System design – it is partitioning the requirements to hardware and software establishing an overall system architecture. This also includes the different processes of the requirements.
- Software design – it represents the software system functions in a form that can be transformed into one or more executable programs.

An example is Unified Modeling Language (UML). There are tools that after you do your object relationship diagram, you can translate that into classes with its properties and methods of which it saves you time to write your object codes.

To clarify, requirements define the function of the system from the client's viewpoint while system and software design describes the system from the software developer's viewpoint.

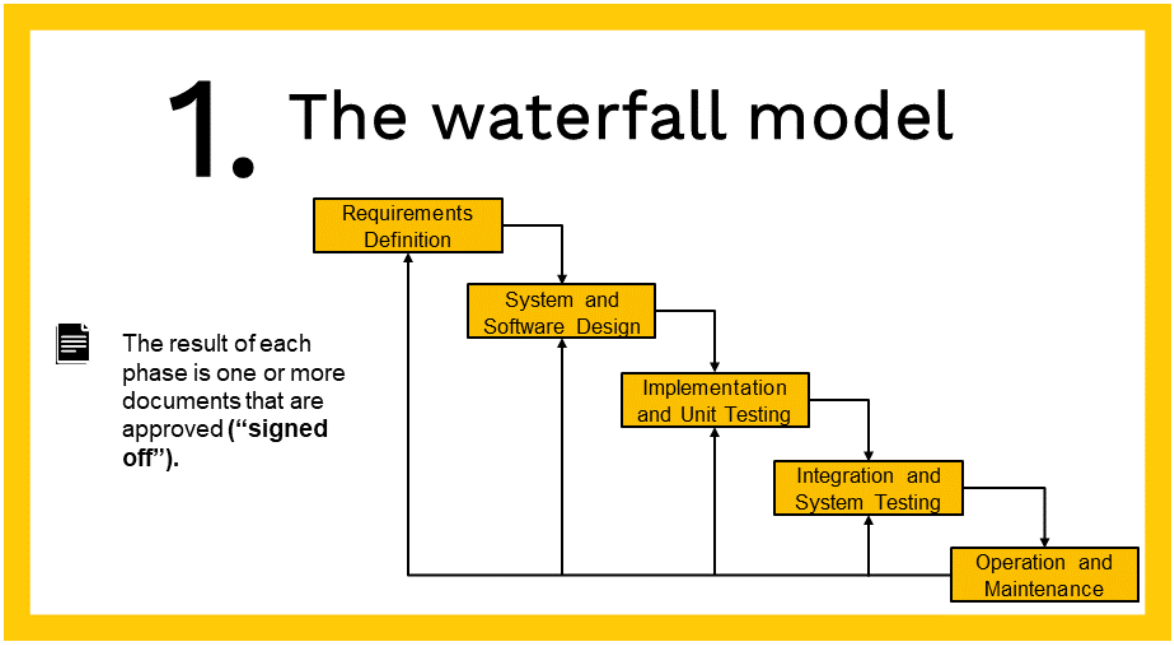
When doing the system and software design, the client's representatives with no background in designing technicality need not be present during the discussion because they will not appreciate it.

Software Process Models

The different software process models included in this lesson are

1. Waterfall
2. Incremental Development
3. Reuse-oriented Software Engineering
4. Prototyping
5. Spiral Model
6. Rational Unified Process

There are more process models but the given are the common ones and good enough to provide you a good start in learning the different process models.



Waterfall model

The first from the list is the waterfall model. It has 5 phases or processes but still aligned with the four fundamental processes.

1. **Requirements specification**
2. **Software design**
3. **Implementation**
4. **Integration and testing**
5. **Operation and maintenance**

If you notice they are synonyms of the four fundamental processes. What makes it famous though is how it performs those processes.

In the Waterfall model, based on the diagram, each phase should not start until the previous phase has finished. To indicate the phase is finished, it needs to have a "signed-off" document.

Phases of the Waterfall Model

1 **Requirements analysis and definition** – the services, constraints, and goals are established in detail and serve as a system specification

2 **System and software design** – identifying the requirements for either hardware or software and have an overall system architecture, system abstraction, and relationship.

This is where your architectural details, UML

knowledge, Entity Relationship Diagram, and data flow diagrams will be used. This is very important because it is in this stage where you will see types of communication protocols your system will be using such as FTP, HTTP, or even TCP/IP among others.

3 **Implementation and unit testing** - This stage is where programmers of software developers turn the design into the program. If you are implementing Object Oriented, part of this stage is to perform unit testing of each object.

4 **Integration and system testing -** Since from the previous stage, the process was to perform UNIT TEST, in this stage you are integrating different processes of the system to see if all requirements are met. As a former developer and some or most may agree, the famous line “that was working a while ago” of which partly true maybe during unit testing.

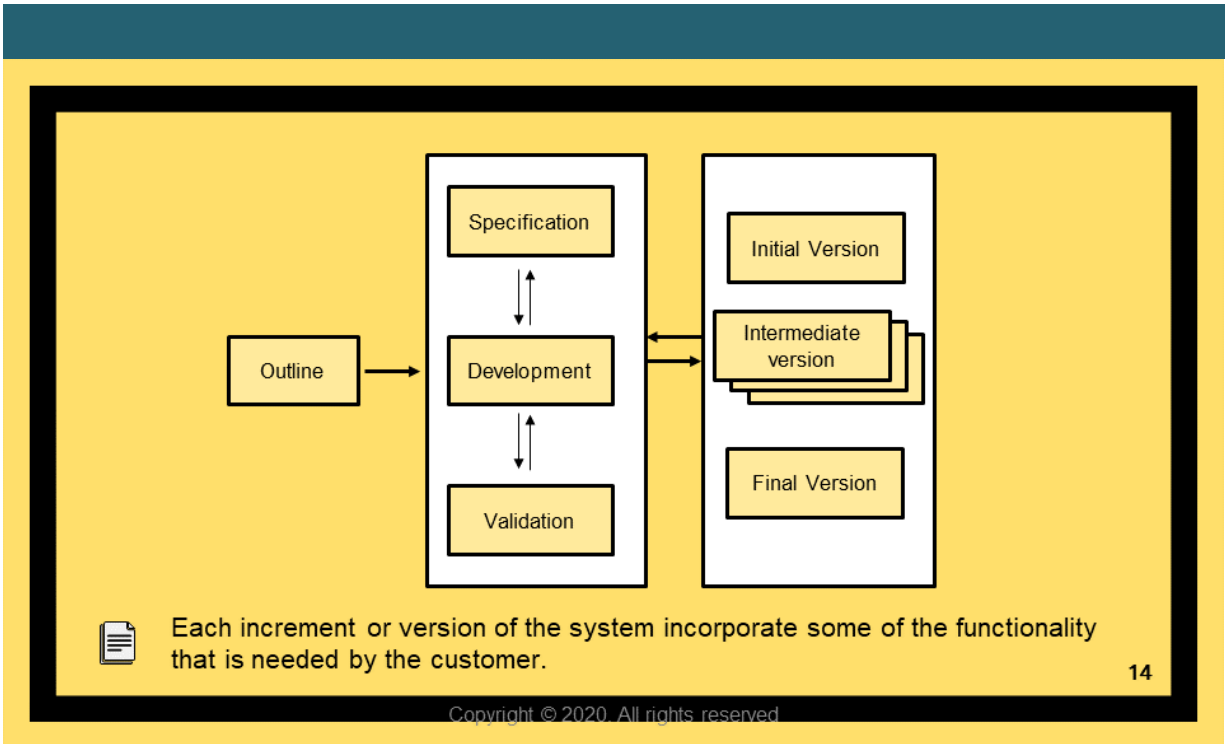
5 **Operation and maintenance** - The system is installed and put into use while at the same time fixing issues that were overlooked during the testing.

Incremental Development

Incremental is based on the idea of developing an initial implementation (with most functionality), exposing it to

users for comments, and evolving it through several versions.

In the industry, we call each release to be phase 1, phase 2, etc… This model is part of the agile development approach.



In this diagram, the process of specification, development, and validation is done in a short period so we can release a version with the prioritized features. Sometimes, a release is scheduled but for some critical issues, it can be released in a form of critical bug fixes.

3 benefits of Incremental Development

1. **Customer change** - The cost of accommodating customer requirements is reduced which means that the software can adapt to changes unlike in waterfall where there is no way of going back to the previous process. In this model, the feedback can be carried over to the next release.
2. **Easier customer feedback** - Customers find it easier to comment on demonstration than that from software design documents. To the clients, understanding the technicality of the document is not a good idea. What they can appreciate is an output where they can comment.
3. **Usefulness of product** - Customers can use and gain value from the software earlier than is possible with a waterfall process.

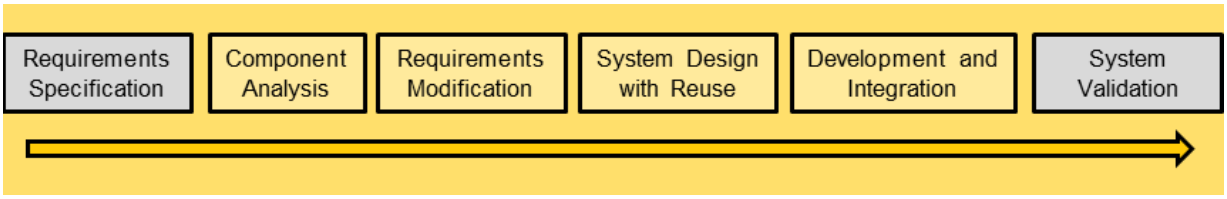
Unlike in waterfall, everything needs to be finished before users can use the system. Sometimes, software projects take years to finish. This results in loss of time and at the same time possibility of going obsolete. Based on experience, projects that took years to finish have a hard time deploying to the users because sometimes, the original stakeholders are already resigned, or sometimes the project turns out to be no longer practical to the current business process.

Reuse-oriented software engineering

This approach is based on the existence of a significant number of reusable components. The process focuses on integrating these components into a system rather than

developing from scratch.

This means that some part of what we are trying to build is already available either paid or free. It is like buying a computer by parts because we want to choose from best based on our budget. The next thing to do is to stitch it together and add the missing parts based on our requirements. This concept is the same as MODULAR.



In the reuse-oriented model, requirements specification and system validation are similar processes from other models such as waterfall and incremental.

Stages of reuse-oriented software engineering

1. **Component analysis** - In this stage, you will find out that when searching for components, usually there is no exact match, therefore only some functionalities are available.
2. **Requirement modification** - Requirements are modified based on requirements specifications. If modification is not available or not possible, a step may go back to component analysis. Sometimes their components that you need to wrap with new properties and methods because it lacks the requirements needed. Sometimes, if this is turning out difficult to do, the option is to go

back to component analysis looking for another candidate component.

3. **System design with reuse** - The framework of the system is designed considering the components that are reused. This means revisiting the design and incorporate the changes to reused components.
4. **Development and integration** - Systems that cannot be externally procured are developed. Components are integrated to create a new system or putting all the pieces.

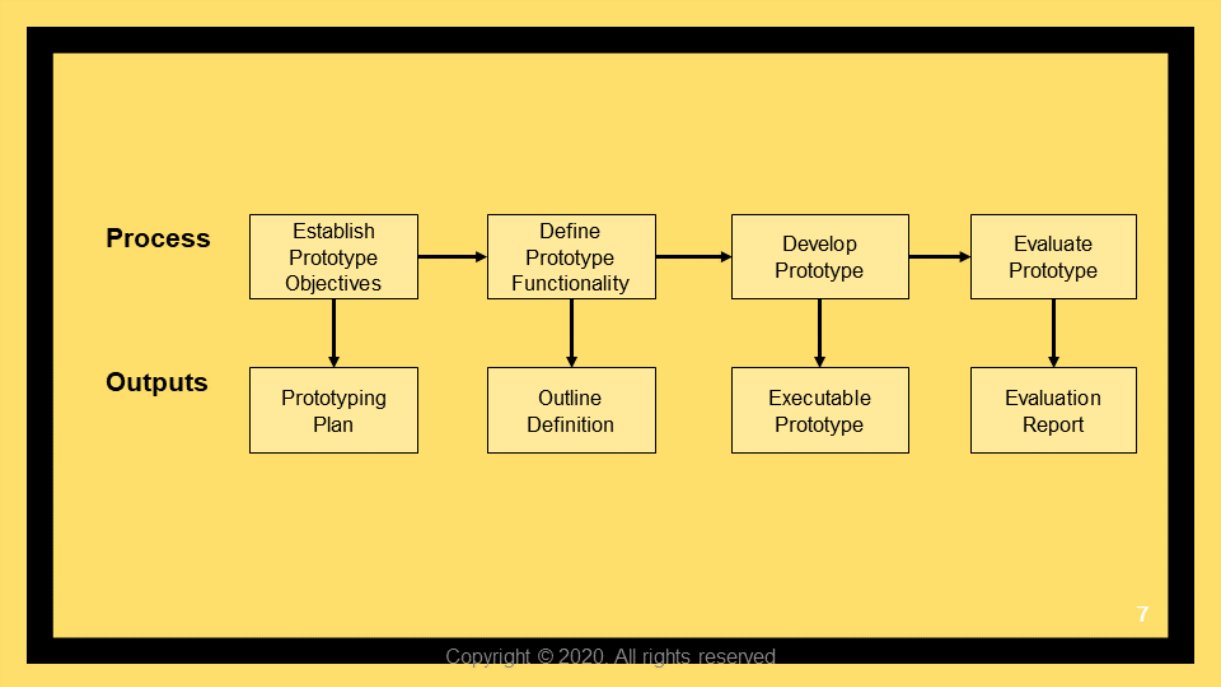
3 types of software components

- **Web services for remote invocation**. This made significant advancement in the reuse-oriented model because it is very flexible and easy to implement.
- **Stand-alone software configured for the environment**. Examples are SDKs to allow the other hardware to communicate with your software.
- **Object collection** - developed as a package to be integrated into frameworks such as .NET or J2EE. Example if the package to manage the database connection.

1.3. Software Processes 2 of 2

Prototyping

- Prototyping is an initial version of a software system that is used to demonstrate concepts, try out
- This is usually used to check the feasibility of a proposed design.



If you check on the process flow, it still is the same with the 4 fundamental activities in software engineering. From system objectives, functionality and so it, it changed to prototype objective, prototype functionality, and others. From the system, it changed to prototype meaning a smaller version of the actual system.

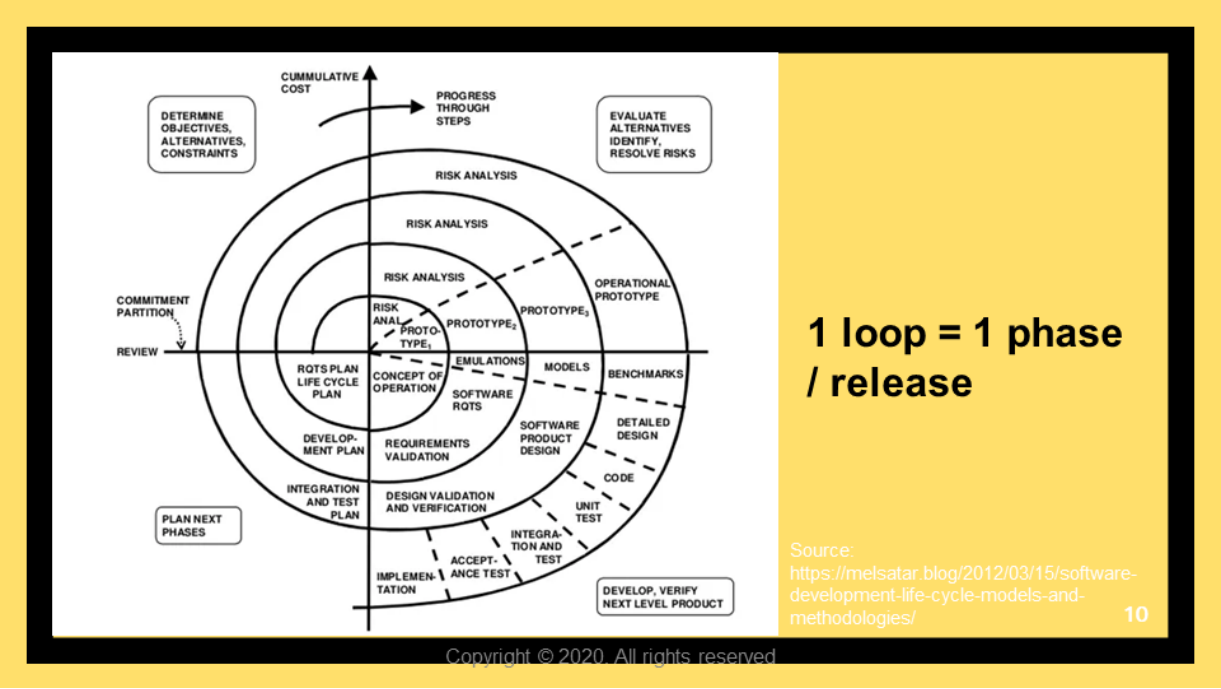
Prototypes do not have to be useful. Paper-based mock-ups of the system user interface can be effective in helping users refine an interface design and work through

scenarios.

Boehm's (1988) Spiral Model

The spiral model's strength and the main difference between other models is its recognition of risks. This allows every loop to incorporate the following risks

- **Project risk** refers to schedule, personnel, fund/cost/ budget
- **Technical risk** refers to implementation, testing, interfaces, maintenance issues. These are the results of ambiguous and incomplete specifications, and technical uncertainty.
- **Business risk**s examples are products that nobody needs, losing monetary funds or personal commitments.
- **Unavoidable risks** examples are technologies discontinued or government policies changing affecting business processes.



Spiral model four sectors

1. Objective setting - As early at this stage, you identify the possible risks and possible alternative same with constraints on the process of the products
2. Risk assessment and reduction - For each risk identified from sector 1, make a detailed analysis and list down the steps to reduce the risks. You can incorporate a prototype to show that risk is reduced or avoided.
3. Development and validation - Perform development and validation by implementing different types of testing and plan and release the iteration.
4. Planning - Review of the project and check remaining items as well as check on the recently released iteration for improvement.

Advantage of the spiral model

1. Risk handling – projects with many unknown risks that occur as the development proceeds, in that case, a spiral is the best development model to follow due to the risk analysis.
2. Good for large projects – since it allows iterative implementation allowing a customer to use the system as early as possible
3. Flexibility in requirements – change request can be incorporated in the next loop or iteration
4. Customer satisfaction – customers use the system in the early stages, no need to wait for the whole system to be completed as practiced by the waterfall.

The disadvantage of the spiral model

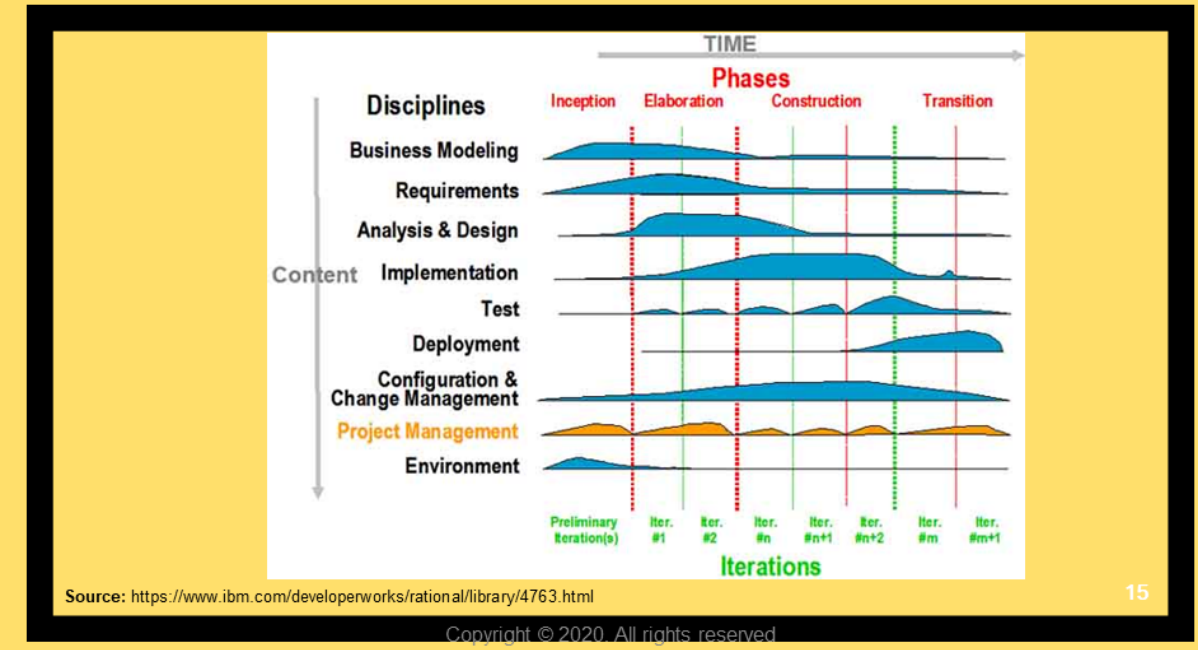
1. Complex – not going to work well on small projects.
2. Expensive – because of the long period of planning and further designing, developing, and implementation
3. Too much dependable on risk analysis – without experienced expertise, it is going to be a failure.
4. Difficulty in time management – because time estimation is difficult and at the same time since it is a spiral, it could turn to infinite.

Rational Unified Process

1 Like other software process models where it is a risk-driven, use-cased, and architecture centric, iterative software development process.

2 Phase model that identifies four discrete phases in the software process namely inception, elaboration, construction, and transition

3 It also combines prototyping and incremental delivery



Based on the diagram, some disciplines are about to start during inception and continues to progress over-elaboration. Some discipline also works throughout phases. Let us check the outputs or results that are rendered for each phase.

Phases of RUP

During **inception**, the outputs are

- The objective statement, initial use case, risk assessment, project plan, prototypes, and models.

During **elaboration**, the outputs are

- Complete use case, architecture, project development plan, prototypes for tackling risks, user manual

During **construction**, the outputs are

- Fully completed software system (this includes initial testing), user manual

During the **transition**, outputs are

- Beta testing, training of users, and rollout or project deployment.

All these outputs are distributed over the disciplines

2.1. Agile Software Development

Introduction

Business operations and transactions have become rapid because of technologies and available opportunities. In these transactions and opportunities, software is part of it. This leads to rapid development of software.

From previous lessons, we introduced some common activities in a software development methodology. Some of them are part of Agile development. In today's lesson, let us check on agile development and how software can be delivered quickly in a small iteration but with usable features added.

Agile methods rely heavily on incremental approach to software specification, development, and delivery. This nature of an ongoing delivery process stems from the need for businesses to adjust to changing requirements and to stay competence in the market. As a result, success in Agile software development is measured by the team's ability to continually deliver.

Agile manifesto

We are uncovering better ways of developing software by doing it and helping

others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Principles of Agile Methods

- **Customer involvement** – customer should be closely involved throughout the development process. Their role is to provide and prioritize new system assignments and to evaluate the iterations of the system
- **Increment delivery** – software is developed in increments with the customer specifying the requirements to be included.
- **People not process** – team members should be left to develop their own ways of working without prescriptive processes.
- **Embrace change** – expect the system requirements to change and so design the system to accommodate these changes.
- **Maintain simplicity** – focus on simplicity in both the software being developed and in the development process. Wherever possible, eliminate complexity from the system.

Types of Agile Method

- Extreme Programming (XP)
- Scrum
- Crystal
- Lean
- Kanban
- Future Driven Development (FDD)

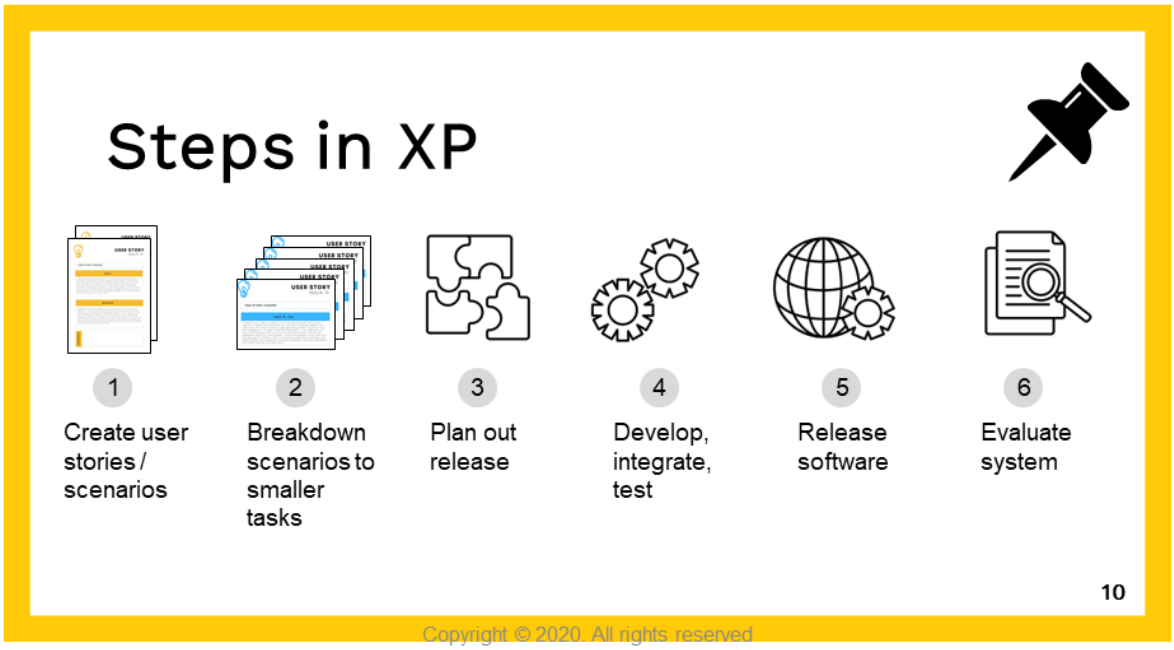
- Dynamic System Development Method (DSDM)

Extreme Programming

“Getting Agile with XP”

- It was developed by pushing recognized good practice, such as iterative development, to “extreme” levels
- Requirements are expressed as scenarios (user stories), which are implemented directly as series of tasks.
- Programmers work in pairs and develop test for each task before writing the code

Steps in Extreme Programming



Take note that after step 6, it goes back step 1 for continuous delivery. Sometimes, as requirements change, the unimplemented stories changed or maybe discarded. If changes are required to stories that have been delivered, new stories will be created and goes over the cycle again.

Spike – an increment where no programming is done. This happens when the team needs to carry out prototyping or trial development to understand problems and solution.

Testing – all test must be performed on old and new functionality. The new build of the system will only be accepted if all tests are executed successfully.

General Problem

Software structure degradation

Changes to the software becomes harder to implement that is why development teams tend to find a work around to problems. This results sometimes to codes duplicated, parts of the software are reused in inappropriate ways, and the overall structure degrades as code is added to the system.

Pair Programming

1. Collective ownership and responsibility for the system
2. Informal review process because code is assessed by two people
3. Helps support refactoring as process of software improvement

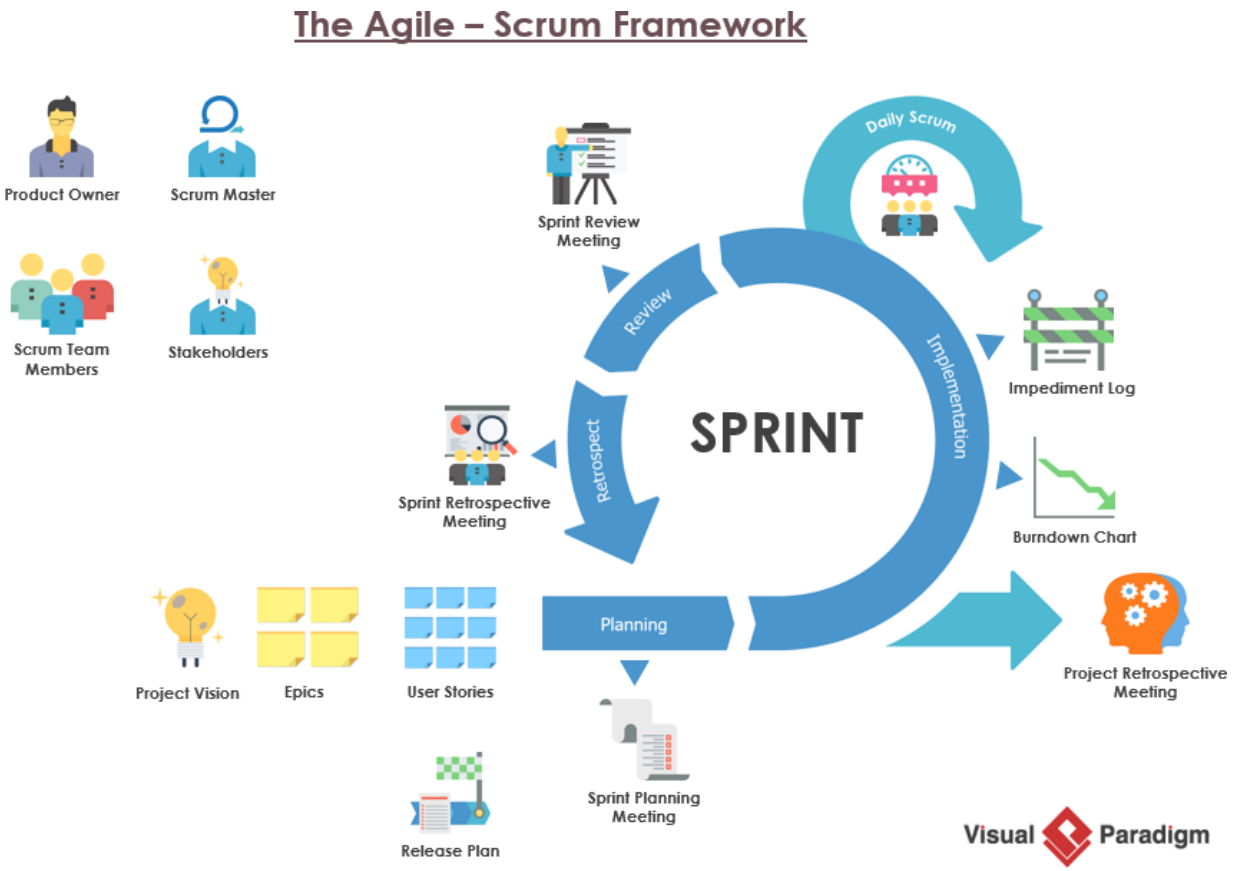
Scrum

“Getting Agile with Scrum”

A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.

Scrum is a framework for developing and maintaining complex products through “view-and-tune”. It is a genre that follows the agile declaration and principles, integrating

three roles, three artifacts, five events, and five values.



3 Roles, 3 Artifacts, 5 Events, 5 Values of Scrum

3 Roles

Product Owner

Development Team

Scrum Master

3 Artifacts

Product Backlog

Sprint Backlog

Product Increment

5 Events

Sprint

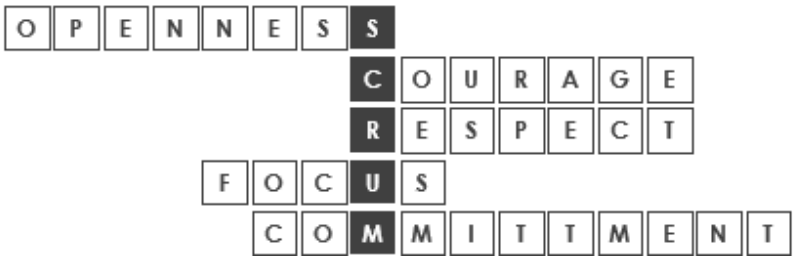
Sprint Planning Meeting

Daily Scrum Meeting

Scrum Review Meeting

Scrum Retrospective Meeting

5 Values



		communicate that vision to the Scrum team. 2 Should understand the business objectives within a wider framework of the market, customer needs, competitors, and digital trends. 3 The product owner not only decides what goes into the Product Backlog but also decides on the priority of each item.
3 Roles		**2 Scrum master** – facilitator of the agile development team. 1 Someone who is expert at Scrum and can therefore coach others 2 Responsible for improving interactions between scrum team and the organization 3 Responsible in

	arranging and facilitating team's daily scrum, planning sessions, and sprint retrospectives. 3 **Development Team** 1 Software engineer, architects, programmers, analyst, system admin, QA experts, testers, UI designers

3 Artifacts	1. **Product backlog**
	    1. Sorted list of all the products needed. Contents comes from product owner
	    2. List of all features, use cases, user stories, improvements, and bug fixes
	    3. Details are descriptions, sequences, and estimated characteristics
	2. **Sprint backlog** – sprint to do list
	    1. Set of product backlog items selected for current sprint
	    2. Contains list of items where development team expects to do on the next increment
	3. **Product increment** – potential shippable product increment

	1. **Sprint** 1. Time boxed

		iteration of a continuous development cycle 2 Planned amount of work must be completed by the team and made ready for review. **2 Sprint planning meeting** 1 Conducted before the start of a sprint in determining the sprint plan and set a sprint goal 2 Includes agreeing on the number of backlogs 3 Product owner describes the priority features to the entire team 4 Whole team should attend (see the 3 roles) **3 Daily scrum meeting** 1 Also called daily stand up meeting usually in the morning on each sprint within 15 minutes

5 Events		2 Discuss on the following 1 What each person did yesterday and what they are going to do today 2 If there are any obstacle 3 If they need assistance 3 Both development team and scrum master need to attend **4 Sprint review meeting** 1 An informal meeting with the development team, scrum master, product owner, and stakeholder 2 Demo on the product and determine what are finished and not finished 3 This is to show to the stakeholder what has been

	accomplished 5 **Sprint retrospective meeting** 1 Occurs after sprint review and before the next sprint planning 2 At most 3 hours meeting for one-month sprints 3 Attendees: all 4 Agenda: 1 What went well in the sprint? 2 What went wrong in the sprint? 3 What we had learn in the sprint? 4 What should we do differently in the next sprint?
5 Values	Open, respect, courage, focus, commitment

2.2. Functional and non-functional requirements

Introduction

In the software industry, there are different types or levels of requirements needed depending on the different kinds of readers. In general, there are **two types of requirements** namely

1. The **user requirements** specifying what the system is expected to be doing, however, elaborated in natural language
2. And the **systems requirements** are containing the more detailed specifications of the system.

One main reason of software failure is due to incorrect documentation. To minimize these failures, we will be studying the different types of requirements and capture what the system should be doing.

Functional and Non-functional Requirements

Functional Requirements – these are statements of services the system should provide, how the system should react to inputs, and how the system should behave situations.

Example of functional requirement from a patient monitoring system

1. A user shall be able to search the appointments lists for all clinics
2. The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day
3. Each staff member using the system shall be uniquely identified by his or her eight-digit employee number

Non-functional requirements – these are the constraints on the services or the functions offered by the system. They include timing constraints, development process constraints, and constraints imposed by the standards.

1. Non-functional requirements may affect the overall architecture of a system rather than individual components.
2. A single non-functional requirement may generate several related functional requirements.

Example: security requirement may turn out to be functional requirements via the following

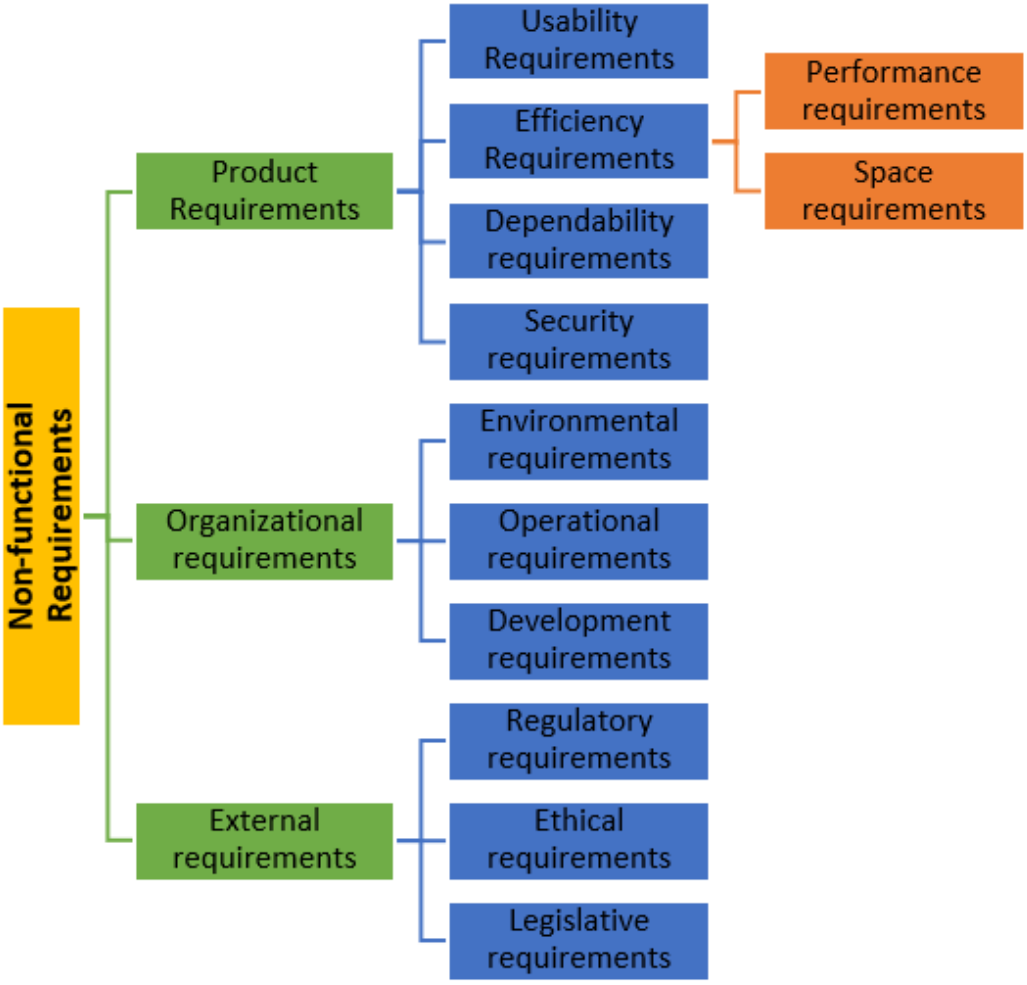
- Two-factor authentications
- Centralized logging
- Encryption of data and API end points

Parameters	Functional Requirement	Non-Functional Requirement
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case.	It is captured as a quality attribute.
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Area of focus	Focuses on user requirement	Concentrates on the user's expectation and experience.
Documentation	Describe what the product does	Describes how the product works
Product Info	Product Features	Product Properties

The difference between functional and non-functional as presented in a table (Dubrova, n.d.)

Types of Non-Functional Requirements

The following summarizes the non-functional requirements (Sommerville, 2011)



Product Requirements	Organizational Requirements	External Requirements

Definition	Specify the behavior of the software	These are broad system requirements derived from policies and procedures in the customer’s and the developer’s organization.	Covers all requirements that are derived from factors external to the system and its development process.

Example	How fast must the system execute? How much memory does it require? Security requirements and others	You are specifying the operational process of how to use the system. You are specifying the programming language and database vendor. On what environment the software must be running	What must be done for the system to be approved by the regulator (central bank, BIR)? Legislative requirements following operations as mandated by law (Data Privacy Act)

Depending on the requirements of the software, not all functional requirements may require. Here are some of the most documented non-functional requirements

- Speed -Processed transactions/

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

2.3. Requirements Elicitation

Introduction

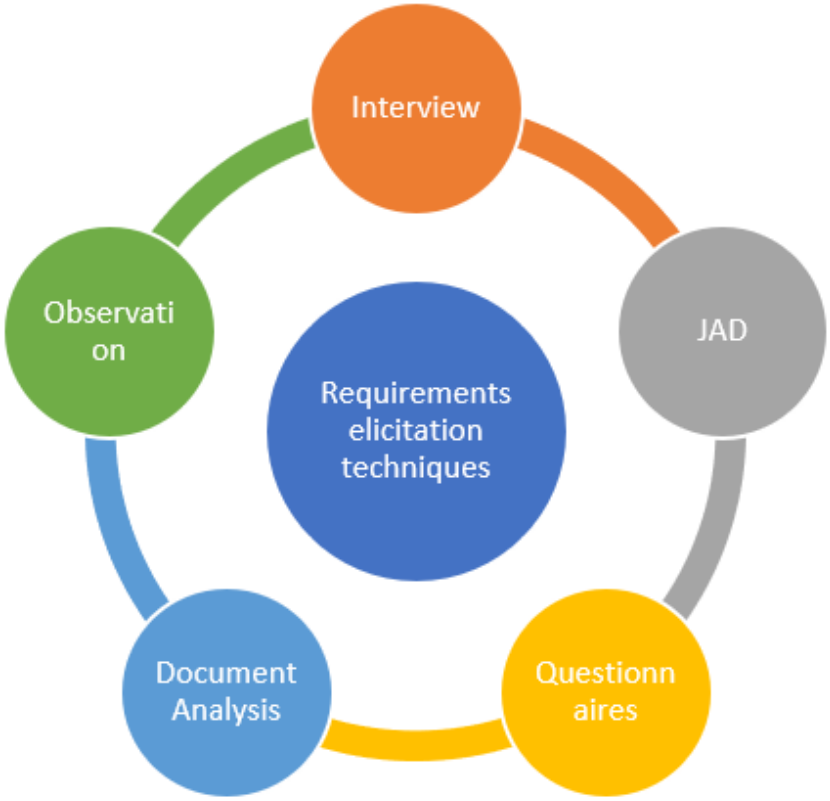
A requirement is a statement of what the system must do or what characteristics (features) it needs to have. It refers to the combination of the following

- Business needs or **business requirements**
- User requirements or **user needs**
- What software should be doing or **the functional requirements**
- What characteristics the software should have or **the non-functional requirements**
- And, how the system should be built or the **system requirements**

Determining requirements

The best source of information when building an information system is the people who would be directly using it. One of the first tasks in determining requirements is to identify the sources of information such as project sponsor, project champion, and all user groups (both direct and indirect).

Requirements elicitation techniques



Interview

The most used requirement elicitation technique. In general, interviews are conducted one on one; however, due to time constraints, several people are interviewed at

the same time. So, how do you perform interviews

1. Selecting interviewees and scheduling interviews. List down the names, position, and your objective as well as the date and time when the schedule will be conducted. You may create a table of interviewees for this.
2. Designing interview questions. There are three types of interview questions

	Type of question	**Examples**
	Closed-ended	How many telephone orders are received per day?
		How do customers place orders?

Open-ended	What do you think about the way invoices are currently processed? What are some of the problems you face daily? Why?
Probing questions	Can you give me an example? Can you explain that in more detail?

3 Preparing for the interview. You should list down the questions you needed to ask in order or category. Confirm the areas of the knowledge area of the interviewee to avoid asking the wrong question. Make sure that when you request a schedule for an interview, send right away the objective so the interviewee can also prepare. Sometimes, it is best to address the

questionnaires beforehand.

4 Conducting the interview. Build rapport, trust, and confidence. Explain why you are performing the conversation and make sure to record all details. Do not be shy to ask questions, follow-up questions, especially when you do not know the answer. Before ending the interview, you both review the details to make sure you did not miss some points.
5 Post-interview follow-up. Prepare an interview report describing the information from the interview (minutes of the meeting) and should be written in 48hours. It is likely to be more forgotten when you do it longer.

Joint application development (JAD)

JAD is a structured process in which 10 – 20 users meet under the direction of a facilitator skilled in JAD techniques. A **facilitator** is a person who sets the meeting agenda and guides the discussion but does not join in as a participant. The following are the steps present in JAD

1 Selecting participants
2 Designing the JAD session
3 Preparing for the JAD session
4 Conducting the JAD session
5 Post-JAD follow-up

Questionnaires

A questionnaire is a set of written questions for obtaining information from individuals. We use it when there are many people involved from whom the information and opinions are needed. We print it or a more significant via electronic such as online forms. The steps are

1. Select participants. Use sampling, so not all people will answer the questionnaire.
2. Designing a questionnaire. Questionnaires must be very clear to avoid misunderstanding. Group related questions and have someone review your inquiries before sending it to participants.
3. Administering the questionnaire. The biggest challenge is how to get the participants to send back the answers. You may indicate a date for submission, or you may personally contact participants to remind them of the deadline.
4. Questionnaire follow-up. Create a report soon after the questionnaire deadline to ensure that the analysis will still be on time.

Document analysis

One way of understanding the “as-is”/existing system is to analyze the documents. The documents included are

- Paper reports
- Memorandums
- Policy manuals
- User training manuals
- Organization chart and forms
- Problems/issues report forms

If you are lucky, you will find excellent technical documentation of the existing system; however, this piece of documents often is missing, especially for systems running over many years.

A good analyst can also find what needs to be changed in an existing system because some reports may never use, and boxes or questions on forms are never or seldomly filled. The most convincing indication that the system needs to be changed is when users create their forms or add additional information to existing ones.

Observation

Observation is the act of watching processes being performed and is a powerful tool in understanding the “as-is” system. It gives the analyst the reality rather than listening to stories and descriptions during interviews or JAD sessions.

Selecting the appropriate techniques

	Interview	JAD	Questionnaire	Document analysis	Observation

Type of information	As-is, improvements, to-be	As-is, improvements, to-be	As-is, improvements	As-is	As-is
Depth of information	High	High	Medium	Low	Low
Breadth of information	Low	Medium	High	High	Low

Integration of information	Low	High	Low	Low	Low
User involvement	Medium	High	Low	Low	Low
Cost	Medium	Low-medium	Low	Low	Low-medium

Depth refers to how rich the information you wanted to collect, and breadth is the range of information.

Requirements Analysis Strategies

Problem analysis

Asking the users and managers to identify the problems with the as-is system and describe how to solve them with the to-be system

Root cause analysis

It focuses on the problems first rather than the solutions. List down all of the issues, analyze the issues, group them, and find the root cause of the issues.

Duration analysis

Duration analysis requires a detailed examination of the **amount of time** it takes to perform each process in the current as-is system. In here, you recreate a scenario

with different methods and set time on each process to see the fastest way.

Activity-based costing

Unlike duration analysis where we measure time, here in activity-based costing, we estimate the cost associated with each process. Assigning value is simple; you just examine the direct cost of labor and materials for each input.

Informal benchmarking

Benchmarking refers to studying how other organizations perform a business process to learn how your organization can do something better.

Outcome analysis

Outcome analysis focuses on understanding the primary outcomes that provide value to customers. One strategy on this is pretending to be directly affected by a problem, so you list down possible results.

Technology analysis

Technology analysis starts by having the analyst and managers develop a list of essential and exciting technologies. Then the group systematically identifies how each technology could be applied to the business process and determines how the business would benefit it.

Activity elimination

The analyst and managers work together to identify how the organization could eliminate each activity in the business process, how the function could operate without

it, and what effects could likely occur.