Final Exam

Compare and contrast the requirements analysis and specification process in the waterfall software development model vs. the Agile software development model.

The **Waterfall** process model was the first of its kind to be produced. It is a linear sequential life cycle model that breaks down project activities into phases. Each phase depends on the previous one as there are no changes once it has passed the given phase. The Waterfall model is typically made up of five main phases, these phases are requirements, design, implementation, verification, and maintenance (There are other depictions of the waterfall model phases but it may depend on the place of work or which resource it was found from). The waterfall model leaves no room for ambiguity as there is a lot of documentation to clearly and specifically defined requirements and what is to come from the given project.

Advantages:

- Easy to understand/use
- Easy to manage, each phase has specific deliverables.
- One phase at a time.
- Works well with small projects.
- Easy to arrange tasks.
- Process and results are well documented.
- Document intensive produces SRS.

Disadvantages:

- No verifiable software is produced until later in the life cycle.
- High risk due to uncertainty.
- Bad for long projects that may need change as they go.
- May be difficult to measure phases.
- Cannot change requirements everything is static.

The **Agile** process model is an iterative process that focuses on individuals interacting repeatedly to produce the correct working software. The model tends to have no or minimal formal documents as repeated validation/specification of requirements maintains the quality of the product. Usually, you use User Stories to elicit/document requirements, these are then used in sprints to produce working features that are then validated or changed through the process. Many models/methodologies apply agile such as the spiral model, Extreme Programming, XP, Scrum, Lean, and Kaban to produce/verify that the correct product is being built.

Advantages:

- The customer is always in the loop, as repeated interactions and continuous deliveries are common.
- Software is delivered frequently.
- Daily conversation/cooperation between business-related individuals and developers.
- Adapt's well to change.
- Welcome's late changing requirements.

Disadvantages:

- It may be difficult to evaluate the length or difficulty of the project initially.
- Lack of documentation
- Unclear interaction with the customer could lead to an unwanted product.
- Customers may derail the whole project if they aren't satisfied with the initial visual models.

What are the (4-6) fundamental activities of software development (also stages of the Software Development Life Cycle) and how does each relate to requirements?

The five fundamental activities of software development according to the SWEBOK Guide are software requirements, design, construction, testing, and maintenance. Overall, every fundamental activity is dependent or based on requirements which are discussed in detail below:

- **Software requirements** - This is the most important stage in SDLC as it sets the foundation for the information/plan for what is to become of the project. It focuses on identifying the overall business motive to produce a product and defines requirements that are to be analyzed, specified, validated, and documented to produce an SRS which consists of all the product requirements to be designed and developed during the project life cycle.

- **Software design** - Design is based on the requirements specified in the SRS which could conclude to multiple design approaches for the product to be documented in a Design Document Specification (DDS). This DDS is reviewed by all important stakeholders to assess which is the best design approach for the product. The design should define all architectural modules of the product along with a data flow representation.
- **Software construction** - Again based on the requirements specified in the SR's developers begin developing the product following coding guidelines defined by the organization and programming tools. This stage could be smoother if the design was detailed and proposed in an organized manner.
- **Software testing** - In the testing stage, the product is verified to ensure that it meets the requirements specified in the SRS. During this stage, multiple tools, measures, and techniques are used to detect product defects to report, track, fix and retest, until the product reaches quality standards. This could be the longest and most expensive stage if the other fundamentals activities were not informative or void of verification.
- **Software maintenance** - During the maintenance stage, the maintenance cost, evolution, categories, and issues are identified. This is to ensure customer feedback is correctly/efficiently handled to overcome any issues and provide a successful product until its retirement. Additionally, there may be processes and activities set in place for product maintenance. This stage can be compared to the User acceptance testing (UAT) found in the software requirement stage validation, which has a set of predefined test scenarios to confirm if the product conforms to the business objective.

(REQ.rfd) What are the (4-5) fundamental requirements activities and how are they performed? (In other words, what is done during the specification/requirements stage of the software development life cycle?) (Within your answer, define a requirement and identify requirements characteristics.)

The four fundamental requirements activities according to SWEBOK Guide are elicitation, analysis, specification, and validation. A brief overview of each activity is described below:

- **Eliciting** requirements is the act of getting requirements from your client, it is involved, iterative, and investigative. A good requirement elicitation process consists of effective communication between various stakeholders throughout the Software Development Life Cycle (SDLC) at different points in time. Requirement specialists must facilitate communication between software users/stakeholders and software engineers to avoid unwanted/unused features. A critical element is describing the software being specified and its purpose and prioritizing the deliverables to ensure the customer's needs are satisfied, i.e. forming the project scope. The process of eliciting requirements has multiple sources to provide a framework for managing software requirements.
- **Analyzing** requirements for software projects consists of Requirements Classification, Conceptual Modeling, Architectural Design and Requirements Allocation, Requirements Negotiation, and Formal Analysis. Requirement classification breaks the classification of requirements into several dimensions. Conceptual Modeling develops models of entities from the problem domain, configured to reflect their real-world relationship and dependencies. Architectural design and Requirements allocation identify at which point the requirement process overlaps with software or system design and illustrate how impossible it is to decouple any two tasks. Requirements negotiation concerns resolving problems with requirements where conflicts occur between two stakeholders requiring incompatible features, between requirements and resources, or between functional and non-functional. Formal analysis requires requirements to have a language with formally defined semantics which have two benefits.
- Requirements **specification** is the detailed formulation, which provides a definitive description of a system to develop or validate the system in document form. This process involves the creation of three main documents called system definition document (high-level system requirements are recorded), system requirements document (system requirements are specified), and software requirements document (specifies what the software product is to do and not to do).
- **Validating** requirements ensures that one is working on the right problem by making sure software engineers understand the requirements. Additionally, it is important to verify that the requirements document conforms to company standards for understandability, consistency, and completeness. Requirements can be validated by doing requirement reviews (setting up inspection of requirements with a group), prototyping (tool to validate requirements and elicit new ones), model validation (system communication paths), and acceptance tests (validates that the finished product satisfies the requirements).

(REQ.er) How would you go about eliciting requirements for a software project?

If placed in a perfect scenario for eliciting requirements I would begin by identifying important stakeholders, goals, business rules, the operational environment, and the organizational environment to understand the current situation of the customer (viewpoints) and why the software to be built is needed. Once these preliminary yet important components have been identified I would apply some elicitation techniques to elicit requirement information from all stakeholders, these techniques are interviews, scenarios, prototypes (potentially mockups or wireframes), facilitated meetings, observations, user stories, and other techniques that may be situational. Additionally, throughout this whole process, I would identify keywords that all stakeholders and software engineers can use as a glossary to avoid confusion when reading the documented requirements. Depending on the size of the project the best elicitation technique may differ, when working with smaller teams/projects I would set up multiple facilitated meetings to get everyone involved and ensure everyone's voice is heard. Having these facilitated meetings ensures that no one is out of the loop especially in small teams and it allows the business analyst (or whoever is in charge of eliciting requirements) to use user stories and complete observations easily as the group overall feels comfortable with their presence, for the time being, the last thing you want to do is make the stakeholder/team uncomfortable potentially leaving out important requirements/details that may hurt you in the long run. As for large projects, I believe identifying one major stakeholder or group leader to interview periodically may be the most time-efficient for the given situation. Having too many stakeholders can lead to contradicting requirements that may over-complicate the product leading to something completely unwanted or different from the initially proposed system. Additionally, creating scenarios (conceptual models) and eliciting user stories may be simple to display and gather for a large group to understand the system's flow. One technique I specifically left out for both cases was prototyping as I believe this is one of the most important techniques as you can easily spot misconceptions and elicit new requirements easily (visual models are the easiest way to elicit in my opinion/experience).

(REQ.rsd) What are some documents and diagrams you would create or reference during the requirements engineering process? (include information about the order of creation/reference, target audience, structure, key contents, and benefits)

Documents and diagrams I would create/reference during the requirements engineering process are System Definition Document / Business Requirement Specification (BRS), Stakeholder Requirements Specification (StRS), System Requirements Specification (SyRS), Software Requirements Specification, Use Case Diagrams, and Level 0 Data Flow Diagrams (context diagrams). These documents should be created in the order that they are stated below:

- **System Definition Document** - Defines the high-level system requirements from the domain perspective. Its readership includes representatives of the system users/customers so its content must be couched in terms of the domain. The document lists the system requirements along with background information about the overall objectives for the system, its target environment, and a statement of the constraints, assumptions, and nonfunctional requirements.
- **Business Requirement Specification -** Describes the organization's motivation for why a system is being developed or changed. The BRS content describes how the organization is pursuing new business or changing the current business to fit a new business environment, and how to utilize the system as a means to contribute to the business. It should be created by the business itself, often with a business analyst. The target audience is a business analyst or representative user from the business, business management, system analyst, and software/system engineers.
- **Stakeholder Requirements Specification** - Describes the organization's motivation for why a system is being developed or changed. The StRS content consists of stakeholder requirements such as organizational requirements, business requirements, and user requirements. It should be created and specified by the stakeholders. The target audience is stakeholders and software/system engineers.
- **System Requirements Specification** - This document is created to identify the technical requirements for the system of interest and the envisaged human-system interaction. i.e. the purpose of the SyRS is to describe what the system should do, in terms of the system's interactions or interfaces. The SyRS content includes conceptual models designed to illustrate the system context, usage scenarios, the principal domain entities, data, information, and workflows. It should be created by system or software engineers, with a target audience focusing on the technical community who will build the system.
- **Software Requirements Specification** - Specifies for a particular software product, program, or set of programs that perform certain functions in a specific environment. The SRS content defines all the required capabilities of the specified software product to which it applies, as well as documenting the conditions and constraints under which the software has to perform, and the intended verification approaches for the requirements. It should be created by software engineers but could be a technical writer, a systems architect, or a software programmer. Its target audience is stakeholders, development teams, quality assurance testers, operations, maintenance, and project consultants.
- **Use Case Diagrams** - Depicts a visual model of possible scenarios in which users will interact with the proposed system. This should be produced along with the SyRS but it's situational and according to company policies.
- **Level 0 Data Flow Diagrams** - A level 0 DFD also known as a context diagram is an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows. Used to depict scope and interfaces.

(REQ.rv) How do you validate requirements?

During the requirements SDLC stage, I believe validation of requirements should be an iterative process that is handled as requirements are created. Requirements need to be analyzed after being elicited to detect and resolve conflicts between requirements (conflict resolution), discover the bounds of the software and how it must interact with its organizational and operational environment (could involve prioritization), elaborate system requirements to derive software requirements. Four different validation techniques can be used to validate the requirements and these are requirements reviews, prototyping, model validation, and acceptance testing. I initially like to begin validation of requirements by classifying them, the classification allows for the creation of a checklist that can be updated periodically. This leads me to hold one or multiple requirement reviews to look for errors, mistaken assumptions, lack of clarity, and deviation from standard practice. Using prototypes and models to validate the requirements are great tools that I would apply to every project, as I believe visual models are the best for stakeholders to understand. These two techniques not only help validate but they help elicit new requirements that you may need to take a step back to classify and review with groups again. Finally, the final technique I would apply to ensure the requirements have been validated is acceptance testing, as it places a test on each requirement to ensure that it satisfies the proposed product. Overall to keep it short, I would validate requirements by first classifying them, holding group reviews, creating prototypes and models, and create acceptance tests to ensure each requirement satisfies the proposed product.

(FGCUScholars 2) What is critical thinking? How do you think critically?

I've always thought of critical thinking as the act of evaluating a problem in multiple aspects to come up with the best solution, but this course has taught me that it can be somewhat more complex but encompasses the same general idea. Critical thinking can be defined as "Thinking about thinking to improve thinking". When given a problem I initially try to understand the purpose of the problem, this could be somewhat ambiguous depending on the complexity of the problem but it could lead to multiple different outcomes as I could attempt to find more information (data, facts, observations) or attempt to understand the point of views of others. With the given information or points of view I've found, I can begin to understand why some assumptions have been made for the given problem. This combination of understanding information, points of view, and assumptions allow me to weigh the implications and consequences that the problem is facing by applying different concepts I've learned or need to be learned. By this point, I may either have concluded the given problem or may need to retrace back to fill the void of any misconceptions I made. To keep it simple, using Elder Paul's elements of thought I've set a pattern that allows me to think critically about a given problem below.

Pattern:

- Understand the issue or purpose of the problem to come up with a quick conclusion (rare)
- If no conclusion is set, identify information and different points of view.
- Identify assumptions made by others.
- Evaluate implications and consequences made about the problem by applying learned or need to be learned concepts.
- Identify the best solution, else I return to a step in which more information needs to be gathered.

This pattern can change according to a given problem, but no matter what pattern I use, I always place sustainability and the increased longevity of our planet as the main goals of engineering solutions (ethics and professional responsibilities).