Examination 1 Study Guide CECS 343

***Chapter 2:***

1. **Stakeholder definition-** These are anyone who has a stake in the successful outcome of the project-business managers, end users, software engineers, support people, and so forth.

* *Stakeholders have some vested interest in the outcome of the software project which can be financially, or workplace invested.*

1. **Process flow definition**- Describes how the framework activities and the actions and the tasks that occur within each framework activity are organized with respect to sequence and time.
2. **Positives for using the spiral model**- The **Pros** of the evolutionary model is to develop high-quality software in an iterative or incremental manner. This process provides flexibility, extensibility, and speed of development. The **Challenges** that are for software teams and managers is to establish balance b/w these critical projects and project parameters and customer satisfaction.
3. **Identifying a task set in software engineering**.- Task sets define the actual work that needs to be done to accomplish the objectives of a software engineering action. The task set can vary depending on the size and complexity of the project. For obvious reasons, larger, more complex projects will take more requirements to get complete details of actual work that is done.
4. **Numeric measures or software analytics-** Many software engineers believe that software processes and activities should be assessed using **numeric measures or software analytics**. The progress you have made in a journey toward an effective software process will define the degree to which you can measure improvement in a meaningful way.
5. **Prescriptive models** define a predefined set of process elements and a predictable process of workflow. These models strive for structure and order in software development. These process models are sometimes referred to as “Traditional” process models.

* *Prescriptive models describe a set of process elements*

1. **Linear sequential model definition**- A.k.a the waterfall method suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment.

* The Waterfall method is the oldest paradigm for software engineering.
* A model that works in a sequential method throughout the software cycle but cannot return to any previous phase

1. **Waterfall models weakness**- The greatest weakness to the waterfall method is that it can be difficult to completely determine all aspects of the start of the project which can cause an ill-design or failed end project.
2. **Popularity of prototyping model**- The prototyping model is considered to be popular due to the ability to engage stakeholders in the process of interacting with a working model before the entire project is completed.
3. **Differentiate unified process model**- This model is different from other models at it recognizes the importance of the customers communication and streamlined methods for describing the customer’s view of a system.

***Chapter 3:***

1. **Agility and the cost of change**- The cost of change increases nonlinearly as a project progresses.

* The cost of change can be easy to accommodate early in the project as a software team is gathering requirements. The cost at this time may be minimal and the time required will not adversely affect the outcome of the project. If the group is in the middle of testing or later in the project and the stakeholder requests a major change the cost can escalate quickly because of the time and effort to make the change immediately.

1. **Agility principle** defines 12 principles for those software organizations that want to achieve agility.

* Not every agile process model applies the 12 principles in equal weight and some sometimes choose to ignore the importance of one over the other. However, these principles define an agile spirit that is maintained in each of the process models presented.

1. Scrum teams and artifacts- ***Scrum team*** is a self-organizing interdisciplinary team ***consisting of a product owner, scrum master, and a small development team***. The principle scrum ***artifacts are the product backlog, the sprint backlog, and the code increment***.

* Development proceeds by breaking the project into a series of incremental prototype development period of 2 to 4 weeks in length called sprints.

1. **Spring planning meeting**- Development team works with the product owner and all other stakeholders to develop the items in the product backlog(they rank the items in the backlog by the importance of the owner’s business needs and the complexity of the software engineering tasks required to compete each of them.

* Sometimes the results of planning meetings identify the missing features needed to deliver the required functionality to the end users

1. **Daily Scrum meeting** is a 15-minute scheduled event at the start of each workday to allow team members to synchronize their activities and make plans for the next 24 hours.

* These meetings allow teams to uncover potential problems early and it's the scrum masters task to clear obstacles presented before the next scrum meeting if possible.
  + These are not problem solving meetings, those occur off-line and only involve affected parties

1. **Sprint review meeting** occurs at the end of a sprint when the development team has judged the increment complete.

* The sprint review is often time-boxed as a 4-hour meeting for a 4-week sprint. The scrum master, development team, the product owner, and selected stakeholders attend this meeting.
* Primary activity is a demo of the software increment completed during the sprint.
  + *Demo may not contained all functionalities but rather those functions that were to be delivered within the time-box defined for the sprint*

1. **Sprint Retrospective** - Before the beginning of another sprint planning meeting, the Scrum master will schedule a 3-hour meeting(*for a 4-week sprint)* with the development team called a sprint retrospective to go over the ***what went well in the previous sprint***, ***what could be improved****,* ***what the team will commit to improving in the next sprint***.
2. **XP framework** - Also known as the extreme programming framework is widely used in software development. This **framework encompassess a set of rules and practices** which occur within the context of ***four framework activities: planning, design, coding, and testing.***
3. **Kanban** - This method is a lean methodology that describes methods for improving any process or workflow. Kanban is focused on change management and service delivery. Change management defines the process through which a requested change is integrated into a software-based system.
4. **DeveOPs** attempts to apply agile and lean development principles across the entire software chain supply. DevOps approach involves several stages that loop continuously till the desired product exists***(Continuous Development, Continuous Testing, Continuous Integration, Continuous Deployment, Continuous Monitoring)***

***Chapter 7:***

1. **Requirements engineering** is the term for the broad spectrum of tasks and techniques that lead to an understanding of requirements.

* Requirements engineering is a *major software engineering action that begins during the communication activity and continues into the modeling activity.*
* Requirements engineering establishes a solid base for design and construction.
* Requirements engineering builds a bridge to design and construction.

1. **Requirements elaboration** - Elaboration task focuses on developing a refined requirements model that identifies various aspects of software function, behavior, and information.

* Elaboration is driven by the creation and refinement of user scenarios obtained during elicitation.

1. **Requirements validation** examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected and that all work products conform the standards established for the process, the project, and the product.

* Primary requirements validation mechanism is the technical review

1. **Define stakeholder** - Anyone who benefits in a direct or indirect way from the system which is being developed.

* Each stakeholder has a different view of the system, achieves different benefits when the system is successfully developed, and is open to different risks if the development effort should fail.

1. **Collaborative requirements gathering** goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements.
2. **Developing use case** - Use cases tell a stylized story about how an end user interacts with a system under a specific set of circumstances.

* To develop a use case you must answer the following questions.
* first identify the primary and secondary actor(s).
* Then find what those actor(s) goals are.
* Following that, what preconditions should exist before the story begins.
* What main tasks or functions are performed by the actor?
* What exceptions might be considered as the story is described?
* Are there any variations in the actor’s interaction?
* What system information will the actor acquire, produce, or change?
* Will the actor have to inform the system about changes in the external environment?
* What information does the actor desire from the system?
* Does the actor wish to be informed about unexpected changes?

1. **Analysis model definition** - The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system. The model changes dynamically as you learn more about the system to be built, and as stakeholders understand more about what they really require.
2. **Define secondary actor** - Secondary actors support the system so that the primary actors can do their work.
3. **Requirement negotiation** - are used to develop a projects plans that meets stakeholder needs while at the same time reflecting the real-world constraints that have been placed on the software team.
4. **Requirements monitoring** supports continuous validation by analyzing user goal models against the system in use.

* Requirements monitoring can be extremely useful when incremental development is used. It encompasses 5 tasks:

1. Distributed debugging uncovers errors and determines their cause
2. Run-time verification determines whether software matches its specification
3. Runtime validation assesses whether the evolving software meets the user goals
4. Business activity monitoring evaluates whether a system satisfies business goals
5. Evolution and codesign provides information to stakeholders as the system evolves

***Chapter 8:***

1. **Requirements model objective** - Allows you to elaborate on basic requirements established during the inception, elicitation, and negotiation tasks that are part of requirements engineering.

* Requirements analysis results in the specification of software’s operational characteristics, indicates software’s interface with other system elements, and establishes constraints that software must meet.

1. **UML acronym** - Unified Modeling Language

* Used to support their work
* Contains a robust notation for modeling and development of object-oriented systems and has become a de facto industry standard for modeling software of all types

1. **Requirements model types** - Investigators have identified requirements analysis problems and their causes and have developed a variety of modeling notations and corresponding sets of heuristics to overcome them.
   1. Scenario-based models of requirements- From the POV of various system “actors”
   2. Class-oriented models- represent object-oriented classes and how classes collaborate to achieve system requirements
   3. Behavioral Models- Depict how the software reacts to internal or external “events”
   4. Data Models- Depict the information domain for the problem
   5. Flow Oriented Models- Represent the functional elements of the system and how they transform data as they move through the system
2. **Data models** depict the information domain for the problem
3. **UML actors and profiles**
   1. **UML actor models** are an entity that interacts with a system object. Actors may represent roles played by human stakeholders or external hardware as they interact with system objects by exchanging information.
   2. **UML profile** provides a way of extending an existing model to other domains or platforms. This might allow you to revise the model of a WEb-based system and model the system for various mobile platforms. *Profiles might also be used to model the system from the viewpoints of different users*.
4. **Use case trigger** - The trigger identifies the event or condition that “gets the use case started”.
5. **Potential classes** - Classes are determined by underlining each noun or noun phrase and entering it into a simple table. If the class is required to implement a solution, then it is part of the solution space; otherwise, if a class is necessary only to describe a solution, it is a part of the problem space.
   1. What should we look for once all the nouns have been isolated? Analysis classes manifest themselves in one of the following ways:
      1. External Entities
      2. Things
      3. Occurrences or events
      4. Roles
      5. Organizational units
      6. Places
      7. Structures
6. **Sequence diagrams** - Sequence diagrams can be used for behavior modeling. They also can be used to show how events cause transitions from object to object(sequence diagram is a shorthand version of the use case). It represents key classes and the events that cause behavior to flow from class to class.
7. **State diagrams** in the context of behavioral modeling, two different characterizations of states must be considered: (1) The state of each class as the system performs its function and (2) the state of the system as observed from the outside as the system performs its function.
   1. The state of a class takes on both passive and active characteristics
      1. Passive State- The current values assigned to an objects attributes
      2. Active State- The objects indicates the status of the object as it undergoes a continuing transformation or processing
8. **Activity diagrams** supplement the use case by providing a graphical representation of the flow of interaction within a specific scenario. Software engineers like to describe activity diagrams as a way of representing how a system reacts to internal events.

***LinkedIn Learning Video- Software Design: Developing Effective Requirements***

Review the following topics

1. Requirements Types and Phases What are requirements

* Requirement Types
  1. **Business Requirement:** High level objective
  2. **Business Rule:** Policy, Standard, Regulation
  3. **Functional Requirement:** System behavior
  4. **Nonfunctional Requirement:** System characteristics “-ilities”
  5. **External interface requirement:** System Integration
  6. **Constraint:** Developer Choices
* Phases:
  1. Elicitation
  2. Specification
  3. Analyzation
  4. Validation

2. Requirement Elicitation

Start with a vision

*Software Requirements*

1. Solve a business problem, improve a business process, or exploiting a business opportunity
2. Product vision- long term vision of the project we are trying to build
3. Project Scope- Product vision helps define the project scope

Elicitation techniques

User Involvement

1. *Involve the users with interviews, workshops, focus groups, observations, questionnaires*

Independently

1. *System interface analysis, user interface analysis, document analysis*

Functional vs. non-functional

1. Functional requirements represent what the system should do
   1. *Business process that is going to be automated*
   2. *User tasks, often articulated by the users, business analyst’s focus*
2. Non-functional requirements represents how the functional requirements do their tasks
   1. *Cut across functional requirements*
   2. *System Properties, Often unstated and assumed, architect’s focus*

4. Requirement Validations

Validation techniques

* 1. **Verification:** Check if you are doing something correctly
  2. **Validation:** Check if you are doing the right thing
  3. Early validation significantly reduces the cost of fixing bugs
  4. Templates and Checklist -> Peer Review -> Prototype = Quality
  5. Developing a checklist
     1. Correctness
        1. Requirement correctly captured
        2. Use case not too broad or vague
     2. Completeness
        1. All scenarios Identified
        2. All Actors identified
     3. Readability
        1. Active voice
        2. Domain terminology defined in shared glossary

Class Diagram

* You need to document the methodologies on how you identify the classes and their relationship