Module 1: Inception

- Vision Document: High-level description of the system's purpose and goals.
- Use Case: Describes interactions between actors and the system that yield value.
- Scope Management: Identifying and controlling project requirements to manage risk.
- Business Case: Justification for the project, including financial and non-financial benefits.

Module 2: Elaboration

- Architecture Baseline: A stable structure from which to proceed with detailed design.
- Risk Assessment: Identifying, analyzing, and managing risks.
- Use Case Model: Collection of use cases representing system functionality.
- Iteration: Small cycles of development to gradually refine the system.

Module 3: Requirements Elicitation

- Stakeholders: People with an interest in the system, including users and experts.
- Interviews: Elicitation technique where stakeholders provide requirements.
- Prototyping: Creating a working model to gather feedback on requirements.
- JAD (Joint Application Development): Collaborative workshops for stakeholders to discuss requirements.

Module 4: Requirements Analysis

- Data Models: ER diagrams or class diagrams to represent data structures.
- Behavioral Models: Use case diagrams, sequence diagrams, or statecharts to show system behavior.
- Flow Models: Data flow diagrams to show the flow of data within the system.
- 4+1 View Model: Architecture model with Logical, Process, Deployment, Implementation, and Use Case views.

Module 5: Requirements Specification with Use Cases

- Use Case Diagram: A visual representation of actors and their interactions with the system.
- Happy Day Scenario: The ideal flow where everything works as expected.
- Preconditions: Conditions that must be true for a use case to start.
- Postconditions: Results expected after a use case has been executed.

Module 6: Use Case Analysis

- <<include>>: A relationship where one use case includes the behavior of another.
- <<extend>>: A relationship where a use case can extend another for specialized behavior.
- Generalization: Use cases or actors that inherit behavior from others.
- Analysis Class: A class representing a key abstraction or responsibility within the system.

Module 7: Activity Diagrams

- Action: An atomic step in a workflow.
- Activity: A group of related actions.
- Swimlane: A partition of activities by who performs them (e.g., actors or systems).
- Fork/Join: Represents concurrency in an activity diagram where multiple actions occur simultaneously.

Module 8: Sequence Diagrams

- Lifeline: Represents the lifecycle of an object or actor in a sequence diagram.
- Synchronous Message: A message where the sender waits for a response.
- Asynchronous Message: A message where the sender does not wait for a response.
- Loop: A repeated interaction between objects.

Module 9: Requirements Quality

- Correctness: Ensuring all requirements are accurate and meet user needs.
- Completeness: All necessary requirements are included.
- Unambiguous: Requirements can only be interpreted one way.

- Verifiability: A requirement is testable to ensure it's implemented correctly.
- Traceability: Ensures each requirement can be traced to design, code, and tests.

Module 10: Requirements Standards

- IEEE 29148: Modern IEEE standard for software requirements, covering process, elicitation, and management.
- CMMI: Capability Maturity Model Integration, a process improvement framework that includes requirements development and management.
- SWEBOK: Software Engineering Body of Knowledge, a comprehensive guide to software development, including requirements engineering.
- DO-178C: Avionics software standard that mandates rigorous traceability and verification.

Module 11: Requirements Decomposition

- Flow-Down: Assigning high-level requirements to subsystems.
- Refinement: Breaking down requirements into actionable details for implementation.
- Completion: Ensuring all code is traced to requirements, particularly in safety-critical industries.
- Derived Requirements: Requirements inferred or added during refinement or decomposition.

Module 12: Requirements Management

- Traceability: Linking requirements to project elements like design, tests, and code.
- Change Control Board (CCB): A committee that manages and approves changes to requirements.
- Change Request Management (CRM): The process of handling changes to requirements.
- Requirements Maturity Levels: Stages of requirements management from chaos (no management) to integrated (fully integrated requirements management).

1. Business Domain Modeling

- Purpose: Understand the problem's broader context (organization, domain, and process improvements)(Business Domain Modelin...).
- Approaches:
    - Eriksson & Penker: Business vision, process, structure, and behavior (Business Domain Modelin...).
    - Jacobsen: Business use case modeling with UML(Business Domain Modelin...).
- Key Focus:
    - Define a common vocabulary early to prevent misunderstandings(Business Domain Modelin...).
    - Compare business models (specific to an organization) vs. domain models (independent abstractions for reuse across applications)(Business Domain Modelin...).

2. Unified Process (RUP) Phases

- Inception (What to Build):
    - Focus: Vision, high-level requirements, and business case(Module 1 - Part 1 - Inc...).
    - Key Deliverables: Vision document, initial use case catalog(Module 1 - Part 1 - Inc...).
    - Scope Management: Reduce risk by identifying key requirements and managing changes(Module 1 - Part 1 - Inc...).
- Elaboration (How to Build):
    - Focus: Detailed requirements (~80%), stable architecture(Module 1 - Part 2 - Ela...).
    - Key Deliverables: More complete use case catalog, architecture baseline (Module 1 - Part 2 - Ela...).
    - Address risks: Business, technical, team, and tool-oriented risks(Module 1 - Part 2 - Ela...).

3. Needs, Features, and Requirements

- Needs:
    - Reflections of business or operational problems(Module 2 - Part 1 - Nee...).
    - Can be vague; understanding them helps define the true nature of the problem(Module 2 - Part 1 - Nee...).
- Features:
    - High-level system services to fulfill stakeholder needs(Module 2 - Part 1 - Nee...).
    - Features are identifiable, but not directly implementable(Module 2 - Part 1 - Nee...).
- Problem Analysis Heuristics:
    - Agreement on problem definition(Module 2 - Part 1 - Nee...).
    - Understanding root causes through techniques like the 5 Whys(Module 2 - Part 1 - Nee...).
    - Identify stakeholders, end-users, and system constraints(Module 2 - Part 1 - Nee...).

4. Key Tools and Techniques

- Traceability: Linking needs to features and requirements(Module 2 - Part 1 - Nee...).
- Root Cause Analysis: Techniques like fishbone diagrams and Pareto charts to identify underlying issues(Module 2 - Part 1 - Nee...).

5. Common Pitfalls to Avoid

- Inception: Avoid too much formality and analysis paralysis(Module 1 - Part 1 - Inc...).
- Elaboration: Time-box the work to avoid perfectionism and scope creep(Module 1 - Part 2 - Ela...).

Module 2: Requirements Elicitation

1. What is Elicitation?
    - Process of drawing out latent or implicit knowledge from stakeholders (Module 3 - ReqsElicitat...).
    - Involves making explicit what is known but not clearly stated.
2. Key Players in Elicitation
    - Users: End-users of the software (can also be other systems).
    - Buyers: Those responsible for purchasing the software (may differ from users).
    - Experts: Domain experts who offer specialized knowledge(Module 3 - ReqsElicitat...).
3. Information Sources
    - People (users, buyers, experts), documents, reference works, and informal channels like conversations and forums(Module 3 - ReqsElicitat...).
4. Elicitation Techniques
    - Interviews: Structured/unstructured, rich information, but time-intensive.
    - Group Meetings: Semi-structured workshops, good for consensus building.
    - Storyboarding/Prototyping: Create visual representations for feedback.
    - Questionnaires: Good for reaching a broad audience, but may have poor response rates.
    - Observation (Ethnography): Time-consuming but reveals natural user behavior(Module 3 - ReqsElicitat...)(Module 4 - Requirements...).
    - Joint Application Development (JAD): Collaborative workshops that include multiple stakeholders(Module 3 - ReqsElicitat...).
5. Common Issues in Elicitation
    - "Yes but...": Users continually adding features.
    - "Undiscovered Ruins": New requirements emerging during elicitation.

- User-Developer Communication Gaps: Addressed by multiple techniques (Module 3 - ReqsElicitat...).

Module 2: Requirements Analysis

1. Purpose of Requirements Analysis
    - Transform "what" (user needs) into "how" (design)(Module 4 - Requirements...).
    - Focus on creating a communicative model between stakeholders and developers.
2. Requirements Baseline
    - A formalized document (SRS) that states system functionality and constraints(Module 4 - Requirements...).
    - Acts as the foundation for the design and implementation phases.
3. Analysis Techniques
    - Data Models: ER diagrams, object-oriented analysis to represent data relationships(Module 4 - Requirements...).
    - Behavioral Models: Use cases, statecharts, and sequence diagrams to represent system behaviors(Module 4 - Requirements...).
    - Flow Models: Data flow diagrams (DFD), process models, and activity diagrams to illustrate functional flows(Module 4 - Requirements...).
4. Architectural Methods
    - Top-down approach: Starting from high-level analysis and breaking it down.
    - Bottom-up approach: Building solutions to smaller problems and integrating them later(Module 4 - Requirements...).
    - Leveraging legacy systems: Incorporating existing components into new designs(Module 4 - Requirements...).
5. Key Concepts
    - RUP’s 4+1 View Model: Logical view, process view, deployment view, implementation view, and use-case view(Module 4 - Requirements...).
    - Multiple analysis models may be required to express different perspectives.

Study Focus:

- Elicitation: Techniques, key players, and common issues.
- Analysis: Transition from requirements to design, various modeling methods, and architectural approaches.

Module 3: Requirements Specification with Use Cases

1. What is a Use Case?
    - Describes sequences of events between an actor and a system that yield a valuable result(Module 5 - ReqsUseCases...).
    - Focuses on system behavior, capturing how a system acts and reacts (scenarios)(Module 5 - ReqsUseCases...).
    - Each use case is a template for related scenarios (Happy Day case and variants)(Module 5 - ReqsUseCases...).
2. Components of a Use Case Model
    - Use Case Diagram: Visual representation showing actors, use cases, and their relationships(Module 5 - ReqsUseCases...).
    - Use Case Specification: Textual description detailing the use case goal, actors, trigger events, and pre/postconditions(Module 5 - ReqsUseCases...).
    - Glossary: Defines specific terminology used in the system (e.g., FURPS: Functionality, Usability, Reliability, Performance, Supportability)(Module 5 - ReqsUseCases...).
3. Use Case Development Process
    - Step 1: Identify and describe actors.
    - Step 2: Identify use cases and write a brief description(Module 5 - ReqsUseCases...).
    - Step 3: Identify relationships between actors and use cases.
    - Step 4: Outline the individual use cases, focusing on the main success scenario(Module 5 - ReqsUseCases...).
    - Step 5: Refine use cases by identifying alternate scenarios and failure cases.
    - Step 6: Validate use cases with stakeholders(Module 5 - ReqsUseCases...).
4. Best Practices and Pitfalls
    - Avoid functional decomposition (breaking a system into isolated tasks) (Module 5 - ReqsUseCases...).

- Focus on actor goals and how use cases satisfy them(Module 5 - ReqsUseCases...).
- Ensure use cases deliver a valuable result and are not overly granular (e.g., "Insert Card" is too small)(Module 5 - ReqsUseCases...).

Module 3: Use Case Analysis

1. Purpose of Use Case Analysis
    - Focuses on understanding how use cases relate to one another after they have been independently developed(Module 6 - Use Case Ana...).
    - Two use case models are suggested: one after elicitation and another at the start of analysis(Module 6 - Use Case Ana...).
2. Use Case Relationships
    - <<include>>: Behavior of one use case is included in another. Example: Authentication must include fingerprint verification(Module 6 - Use Case Ana...).
    - <<extend>>: Behavior of one use case may be extended by another use case. Example: Additional logging after 5 PM(Module 6 - Use Case Ana...).
    - Generalization: One use case inherits and extends the behavior of another (Module 6 - Use Case Ana...).
3. Steps in Use Case Analysis
    - Step 1: Identify shared behaviors and refactor into new use cases(Module 6 - Use Case Ana...).
    - Step 2: Promote visibility of important extensions in the use case diagram (Module 6 - Use Case Ana...).
    - Step 3: Consider special cases or specialized actors(Module 6 - Use Case Ana...).
    - Step 4: Partition behaviors into analysis classes (boundary, entity, control classes)(Module 6 - Use Case Ana...).
    - Step 5: Begin thinking about high-level architecture(Module 6 - Use Case Ana...).

4. Classes in Use Case Analysis
    - Boundary Classes: Interface with the system (e.g., UI, system interfaces) (Module 6 - Use Case Ana...).
    - Entity Classes: Represent key abstractions in the system (e.g., data)(Module 6 - Use Case Ana...).
    - Control Classes: Coordinate behavior across multiple use cases(Module 6 - Use Case Ana...).
5. Pros and Cons of Use Case Modeling
    - Pros:
        - Provides early buy-in from users and domain experts(Module 6 - Use Case Ana...).
        - Helps identify who interacts with the system and what the system does(Module 6 - Use Case Ana...).
    - Cons:
        - Only captures functional requirements(Module 6 - Use Case Ana...).
        - Organizing common functionality can be challenging(Module 6 - Use Case Ana...).

Module 4: Activity Diagrams

1. Purpose of Activity Diagrams
    - Describes the stepwise flow of activities and actions(Module 7 - Activity2023...).
    - Used to model workflows (organizational or computational processes) and flow between or within use cases(Module 7 - Activity2023...).
    - Helps identify preconditions and postconditions of use cases(Module 7 - Activity2023...).
2. Key Elements of Activity Diagrams
    - Actions/Activities: Actions are atomic executable steps; activities can be decomposed and are non-atomic(Module 7 - Activity2023...).
    - Transitions: Show the control flow between actions(Module 7 - Activity2023...).
    - Split/Merge: Represent decision points in the flow where control paths diverge or converge(Module 7 - Activity2023...).
    - Fork/Join: Show concurrency, where multiple actions occur simultaneously (Module 7 - Activity2023...).
    - Swimlanes: Structure the flow by assigning actions to actors (who does what)(Module 7 - Activity2023...).
3. Flow Control in Activity Diagrams
    - Decision Nodes: Used for mutually exclusive conditions (e.g., [if this], [else]) (Module 7 - Activity2023...).
    - Merge Nodes: Reunite alternative control flows(Module 7 - Activity2023...).
    - Fork/Join: Indicates parallel processing (fork for starting multiple paths, join for synchronization)(Module 7 - Activity2023...).
    - Termination Nodes:
        - Activity Final: Ends the entire activity, including all flows(Module 7 - Activity2023...).
        - Flow Final: Ends only the current flow, allowing other flows to continue(Module 7 - Activity2023...).

4. Pros and Cons of Activity Diagrams
    - Pros:
        - Maps use case scenarios directly to actions.
        - Intuitive for procedural programmers and includes constructs for concurrency and task assignment(Module 7 - Activity2023...).
    - Cons:
        - Confusion with statecharts and changing terminology between UML versions(Module 7 - Activity2023...).
        - Limited tool support(Module 7 - Activity2023...).

Module 4: Sequence Diagrams

1. Purpose of Sequence Diagrams
    - Used to model temporal ordering and interaction between actors and systems(Module 8 - Sequence Dia...).
    - Shows the lifetimes of objects and the communication (synchronous or asynchronous) between them(Module 8 - Sequence Dia...).
    - Represents scenarios from use cases, highlighting operations, system events, and sequential ordering of operations(Module 8 - Sequence Dia...).
2. Key Elements of Sequence Diagrams
    - Lifeline Boxes: Represent interacting objects or actors(Module 8 - Sequence Dia...).
    - Messages/Operations:
        - Synchronous: Sender waits for a response before continuing(Module 8 - Sequence Dia...).
        - Asynchronous: Sender continues without waiting for a response (Module 8 - Sequence Dia...).
    - Alternative Paths/Conditions: Show choices in the flow based on conditions(Module 8 - Sequence Dia...).

- Loops/Repetitions: Represent repeated operations within the interaction (Module 8 - Sequence Dia...).

3. When to Use Sequence Diagrams
    - For actor-system interactions, especially during the analysis phase(Module 8 - Sequence Dia...).
    - To show object communication, creation, and destruction during the design phase(Module 8 - Sequence Dia...).
    - Can be used to model the scenarios in use cases, showing communication between actors and systems(Module 8 - Sequence Dia...).
4. Best Practices for Sequence Diagrams
    - Create a sequence diagram for each scenario in a use case(Module 8 - Sequence Dia...).
    - Focus on high-level abstraction and avoid getting lost in unnecessary details(Module 8 - Sequence Dia...).
5. Pros and Cons of Sequence Diagrams
    - Pros: Effective for illustrating the flow of operations and events between actors and systems(Module 8 - Sequence Dia...).
    - Cons: It's easy to over-focus on diagrams, leading to a lack of actual code development(Module 8 - Sequence Dia...).

Module 9: Requirements Quality

1. Quality Measures (IEEE-830)
    - Correctness: All requirements must be met by the software; no extra "nice-to-haves"(Module 9 - Requirements...).
    - Completeness: Describes all necessary functionality, performance, constraints, and interfaces(Module 9 - Requirements...).
    - Unambiguous: Each requirement should only have one interpretation (Module 9 - Requirements...).
    - Consistency: No conflicts among requirements(Module 9 - Requirements...).
    - Ranked: Prioritize by importance and stability(Module 9 - Requirements...).
    - Verifiability: Requirements must be testable(Module 9 - Requirements...).
    - Modifiability: The structure must allow easy changes(Module 9 - Requirements...).
    - Traceability: Clear origins and referable in future development(Module 9 - Requirements...).
2. Key Challenges and Solutions
    - Correctness: Requires peer and customer reviews, traceability(Module 9 - Requirements...).
    - Completeness: Joint reviews with users, prototyping to ensure all functional and non-functional requirements are captured(Module 9 - Requirements...).
    - Unambiguity: Use multi-level reviews, prototypes, and measurable criteria (Module 9 - Requirements...).
    - Consistency: Extensive manual reviews to prevent contradictions(Module 9 - Requirements...).
    - Ranked Requirements: Prioritize based on scope, stability, and importance (Module 9 - Requirements...).
    - Verifiability: Testing-aware wording and process-focused validation(Module 9 - Requirements...).

- Modifiability: Embrace change through tools and a change management process(Module 9 - Requirements...).
- Traceability: Use unique identifiers for all requirements and artifacts(Module 9 - Requirements...).

3. Requirements Validation
    - Validation: Ensures the system meets the user's needs. It's more than just testing; it checks for completeness, ambiguity, and omissions(Module 9 - Requirements...).
    - Prototyping: Helps with validating concepts but must be careful about shortcuts(Module 9 - Requirements...).
    - Requirements-Based Testing: Validates use cases and ensures all scenarios are covered(Module 9 - Requirements...).

Module 10: Requirements Standards

1. General Standards
    - IEEE 29148: Focused on requirements process, documents (SRS, StRS, SyRS), and traceability. An extended and detailed version of IEEE 830(Module 10 - Requirement...).
    - CMMI (Capability Maturity Model Integration): Emphasizes process maturity. Includes Requirements Development (RD) and Requirements Management (REQM), focusing on elicitation, validation, and traceability (Module 10 - Requirement...).
    - SWEBOK: Software Engineering Body of Knowledge, covering all essential knowledge for software engineers, including requirements(Module 10 - Requirement...).
    - SEBOK: Systems Engineering Body of Knowledge, guiding systems design and requirement decomposition(Module 10 - Requirement...).
    - BABOK: Business Analyst Body of Knowledge, focusing on people, collaboration, and communication in requirements processes(Module 10 - Requirement...).
2. Industry-Specific Standards

- MIL-STD-498: Defense standard focused on comprehensive documentation like SRS, design descriptions, and interface descriptions(Module 10 - Requirement...).
- DO-178C: Avionics software standard with design assurance levels based on failure consequences. It requires extensive traceability and verification (Module 10 - Requirement...).
- IEC 62304: Medical device software standard focused on risk management, development, and requirements traceability(Module 10 - Requirement...).
- EN 50128: Railway system standard focused on safety integrity levels and full traceability(Module 10 - Requirement...).

3. Why Standards Matter
    - Standards provide a common approach to requirements engineering, ensuring rigor, traceability, and compliance, particularly in industries with critical systems like defense, aerospace, medical devices, and transportation(Module 10 - Requirement...).

Module 11: Requirements Decomposition

1. Importance of Requirements Decomposition
    - Decomposition is critical in large and critical systems like military, aerospace, and telecom(Module 11 - Requirement...).
    - Software engineering may become more automated, but requirements decomposition remains a human-centric task(Module 11 - Requirement...).
2. Types of Requirements Decomposition
    - Flow-Down: Assigning requirements to appropriate subsystems(Module 11 - Requirement...).
    - Refinement: Adding details and constraints to make requirements actionable by design and implementation teams(Module 11 - Requirement...).
    - Completion: Ensuring all requirements are fully traced back to the code (used in critical industries like aerospace)(Module 11 - Requirement...).
3. Examples of Decomposition
    - Crew Alerting System (CAS): System requirements assigned to subsystems like UI or Information Services(Module 11 - Requirement...).
    - Resource Allocation: Example from telecom where time budgets are allocated to different subsystems to fulfill performance requirements (Module 11 - Requirement...).
4. Key Challenges
    - Derived Requirements: Can either mean requirements traced from a higher level or added requirements not traced back to a specific user need(Module 11 - Requirement...).
    - Terminology Confusion: Different industries (e.g., aerospace vs. military) use different definitions for the same terms like "derived requirements" (Module 11 - Requirement...).
5. Why Decomposition Matters
    - Decomposition provides a comprehensive trace between requirements and system components, ensuring everything is accounted for during delivery, maintenance, and evolution(Module 11 - Requirement...).

Module 12: Requirements Management

1. Key Areas of Requirements Management
    - Traceability: Ensuring requirements can be traced back to their origins and forward to design, implementation, and testing(Module 12 - Requirement...).
    - Planning for Change: Change is inevitable, and requirements management must account for new and changing needs(Module 12 - Requirement...).
    - Methodology: Using a structured approach to manage requirements through their lifecycle(Module 12 - Requirement...).
    - Tools: CASE tools are crucial for managing complexity(Module 12 - Requirement...).
2. Managing Change
    - Change as Risk: Requirements can change due to evolving user needs, business environments, or unexpected discoveries(Module 12 - Requirement...).
    - Five Types of Change (from Harker et al.):
        - Mutable: Changes in the customer's business environment.
        - Emergent: New requirements that evolve as understanding improves.
        - Consequential: Changes requested after users see the system.
        - Adaptive: Changes after users find better ways to use the system.
        - Migration: Supporting current users during rollout(Module 12 - Requirement...).
3. Requirements Management Maturity Model
    - Level 0 (Chaos): No requirements management leads to poor quality and missing functionality(Module 12 - Requirement...).
    - Level 1 (Written): Requirements are documented, forming a contract with customers and the implementation team(Module 12 - Requirement...).
    - Level 2 (Organized): Requirements are identified, persisted, and versioned (Module 12 - Requirement...).

- Level 3 (Structured): Classify requirements, track dependencies, and state priorities(Module 12 - Requirement...).
- Level 4 (Traced): Ensure both upward and downward traceability of requirements(Module 12 - Requirement...).
- Level 5 (Integrated): Fully integrated with the overall project process; no changes happen without a full review(Module 12 - Requirement...).

4. Change Request Management (CRM)
    - Centralized Change Control Board (CCB): Serves as the authority for approving changes, managing risks, and balancing competing interests (Module 12 - Requirement...).
    - Process: Capture change requests, evaluate the broader impact, and ensure controlled modifications(Module 12 - Requirement...).
5. Traceability
    - Ensures quality, impact analysis, and verification by linking requirements to related artifacts like design, tests, and code(Module 12 - Requirement...).
    - Traceability is essential for managing complex projects where changes in one area can have cascading effects(Module 12 - Requirement...).

SER415 Final Exam materials(cumulative)

8 quality
measures for requirements

1. Correct
2. Unambiguous
3. Complete
4. Consistent
5. Prioritized
6. Verifiable
7. Modifiable
8. Traceable

Correctness

An SRS is correct if, and only if, every requirement
stated therein is one that the software shall meet

Unambiguous:

A requirement is unambiguous "if and only if it
has only one interpretation"

Completeness

A set of requirements is complete "if and only if it
describes all significant requirements of concern to the user,
including requirements associated with functionality,
performance, design constraints, attributes, or external
interfaces"

Consistency

A requirement set is consistent "if and only if
no subset of individual requirements described within it
are in conflict with one another"

Prioritized

Requirements ranked by importance
and stability

Verifiability

A requirement is verifiable "if and only if there exists
a finite, cost-effective process with which a person or machine
can determine that the developed software system does
indeed meet the requirement

Modifiable

SRS modifiable "if and only if its structure and
style are such that any changes to the requirements can be
made easily, completely, and consistently, while retaining the
existing structure and style of the set

Traceable

A requirement is traceable "if and only if the origin
of each of its component requirements is clear, and there is a
mechanism that makes it feasible to refer to that requirement
in future development efforts".

Other Quality Measures

1) Clear
2) Concise
3)Cohesive
4)Feasible
5) Managed

TERM

Requirements

DEFINITION

Descriptions of the system SERVICES and CONSTRAINTS that are generated during the requirements engineering process

TERM

Phases of Requirements Engineering

DEFINITION

Elicitation

Analysis

Validation

Change Management

TERM

Requirements Elicitation is sometimes called _____ or _____

DEFINITION

discovery or gathering

TERM

Stakeholders

DEFINITION

End-users, managers, engineers involved in maintainance, domain experts, trade unions, etc

TERM

Requirements Analysis

DEFINITION

Translating requirements expressed as needs into software products.

Provide a model to bridge the chasm between business stakeholders and implementers (e.g. design docs)

Architecture

Higher level design

TERM

Requirements Validation

DEFINITION

Demonstrating that the requirements defined the system the customer really wants

TERM

Objectives with Inception

DEFINITION

1. Understand what to build

2. Identify key requirements

3. Determine at least one potential solution

4. Understand costs, schedule, and risk

5. Understand what process to follow and tools to use

TERM

Objectives with Elaboration

DEFINITION

1. Get a more detailed understanding or requirements

2. Design, Implement, validate and baseline the architecture

3. Mitigate risks, produce more accurate schedule & cost estimates

4. Deployment and Development Environments

TERM

Needs

DEFINITION

Problem or opportunity that must be addressed

TERM

Features

DEFINITION

A service the system provides

Identifiable but not implementable

WHAT not HOW

TERM

5 Heuristics in Problem Analysis

DEFINITION

1. Gain agreement on the problem definition

2. Understand the Root Causes

3. Identify Stakeholders and End Users

4. Define the Solution System Boundary

5. Identify Constraints

TERM

Functional Requirements

DEFINITION

What the system does

TERM

Non-functional Requirements

DEFINITION

How well the system does its thing

Stipulations or constraints on the system

TERM

Types of non-functional requirements

DEFINITION

Product

Organizational

External

TERM

Product requirement

DEFINITION

The reqs we often think of.

Reqs which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, usability, etc.

TERM

Organizational requirement

DEFINITION

Internal Stipulations

Reqs which are a consequence of org policies and procedures.

TERM

External requirement

DEFINITION

External Stipulations

Reqs which arise from factors external to the system and its development process

TERM

User Requirements

DEFINITION

Written for (and often with) customers

Natural language

Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge

TERM

System Requirements

DEFINITION

More detailed specifications

A structured doc setting out detailed descriptions of the system services

TERM

Requirements Elicitation Techniques

DEFINITION

-Individual Interviews

-Group Meetings

-Prototyping

-Questionnaires

-Observation

-Research

TERM

Individual Interviews

DEFINITION

2-way communication process

Time sensitive

Could be user, buyer, or expert

TERM

Group Interviews

DEFINITION

N-way communication

Groups of customers, cross-functional teams, buyers, experts, focus groups etc.

Semi-structured

Cons: Group think

TERM

Prototyping

DEFINITION

A structure for individual or group exploration

Participants are end users

Cons: Could push yourself into a corner early on

TERM

Questionnaires

DEFINITION

1-way communication

Possibly anonymous

Cons: False answers, Answer options that are too limiting, answer options that are too broad

TERM

Observation

DEFINITION

Watch real people in the domain

Ethnography

Cons: Observing can cause behavior to change

TERM

Research

DEFINITION

0-way communication

Finding and reading written info and artifacts

TERM

Use Case

DEFINITION

Describes sequences of events between an actor and a system that yield a result of value to the actor

A template for a collection of related scenarios

TERM

Use Case Specification (Parts)

DEFINITION

Objective

Primary Actor

Trigger

Secondary Actor(s)

Pre/Post Conditions

Scenarios (Success/Failure)

TERM

Actor

DEFINITION

Defines a coherent set of roles that users of an entity can play when interacting with the entity

Stick figure

TERM

A use case should focus on the users _____

DEFINITION

GOAL

(you should avoid functional decomposition)

TERM

Steps to create a use case

DEFINITION

1. Identify and Describe the Actors

2. Identify the Use Cases and write a brief description

3. Identify Actor to Use Case relationships

4. Outline the Individual Use Cases

5. Refine the Use Cases

6. Verify & Validate the Use Cases

TERM

<<include>>

DEFINITION

A stereotype of a dependency

A -> B

The behavior of B is ALWAYS included into A

TERM

<<extend>>

DEFINITION

A stereotype of a dependency

A <- B

A possible extension, behavior of B may be incorporated into A

TERM

Generalization

DEFINITION

B inherits the behavior and communication relationships of A and is allowed to override and extend

B is generally a standalone basic use case

Actors may apply Generalization as well

A <- B

e.g. Student <- Graduate Student

TERM

Analysis Modeling Techniques

DEFINITION

Data/object Models

Behavioral Models

Flow Models

TERM

Data/object Models

DEFINITION

* Entity-Relationship (ER)

OOA&D

Data Dictionaries

TERM

Behavioral Models

DEFINITION

Use Case Models

State Machines

TERM

Flow Models

DEFINITION

Process/workflow Models

* Dataflow Diagrams (DFD)

* Sequence Diagrams

* Activity Diagrams

TERM

Other Requirements Quality Measures

DEFINITION

Clear

Concise

Cohesive

Feasible

Managed

TERM

IEEE 29148

DEFINITION

A newer, longer doc (~100 pages).

Focused on definitions

TERM

SWEBOK

DEFINITION

Software Engineering Body of Knowledge

Focused on descriptions

TERM

DO-178C

DEFINITION

Avionics Software Standard

Based on consequences of failure

Full bidirectional traceability

TERM

Requirements Maturity Levels

DEFINITION

0 - Chaos! No Requirements

1 - Written requirements

2 - Organized

3 - Structured

4 - Traced

5 - Integrated

TERM

Change Request Management

DEFINITION

Single Channel for Approval

Requirements to

Design to

Code to

Test to

Maintenance

TERM

Why is Traceability so important?

DEFINITION

Quality

-Can we determine that the req is validated/verified?

Impact Analysis

-What other reqs are impacted?

-What people are affected?

-What downstream artifacts are affected?

TERM

5 Step CHANGE MANAGEMENT Process

DEFINITION

1. Plan for change

2. Baseline the reqs

3. Change Control Board (CCB)

4. Use a Change Control System

5. Maintain Traceability

TERM

Types of Decomposition

DEFINITION

Flow-down

Refinement

Completion

TERM

Flow-down Decomposition

DEFINITION

- Assigning requirements to appropriate subsystems

-An Architectural effort

TERM

Refinement Decomposition

DEFINITION

-Ensure reqs reach level of specificity where implementation can easily follow

-A Requirements effort

TERM

Completion Decomposition

DEFINITION

-Adding reqs to complete missing back traces from code to reqs

-Design or even implementation effort

TERM

Requirements specify ____ to build, not ____ to build it

DEFINITION

what/how

TERM

SMT-LIB

DEFINITION

well recognized standard for specifying formal constraints to be solved by an automated constraint solver

SER 415 - Midterm

Study online at https://quizlet.com/_3uu77u

Requirements	Descriptions of the system SERVICES and CONSTRAINTS that are generated during the requirements engineering process
Phases of Requirements Engineering	Elicitation Analysis Validation Change Management
Requirements Elicitation is sometimes called _____ or _____	discovery or gathering
Stakeholders	End-users, managers, engineers involved in maintainance, domain experts, trade unions, etc
Requirements Analysis	Translating requirements expressed as needs into software products. Provide a model to bridge the chasm between business stakeholders and implementers (e.g. design docs) Architecture Higher level design
Requirements Validation	Demonstrating that the requirements defined the system the customer really wants
Objectives with Inception	1. Understand what to build 2. Identify key requirements 3. Determine at least one potential solution 4. Understand costs, schedule, and risk 5. Understand what process to follow and tools to use
Objectives with Elaboration	1. Get a more detailed understanding or requirements 2. Design, Implement, validate and baseline the architecture

	3. Mitigate risks, produce more accurate schedule & cost estimates 4. Deployment and Development Environments
Needs	Problem or opportunity that must be addressed
Features	A service the system provides Identifiable but not implementable WHAT not HOW
5 Heuristics in Problem Analysis	1. Gain agreement on the problem definition 2. Understand the Root Causes 3. Identify Stakeholders and End Users 4. Define the Solution System Boundary 5. Identify Constraints
Functional Requirements	What the system does
Non-functional Requirements	How well the system does its thing Stipulations or constraints on the system
Types of non-functional requirements	Product Organizational External
Product requirement	The reqs we often think of. Reqs which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, usability, etc.
Organizational requirement	Internal Stipulations Reqs which are a consequence of org policies and procedures.
	External Stipulations

External requirement	Reqs which arise from factors external to the system and its development process
User Requirements	Written for (and often with) customers Natural language Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge
System Requirements	More detailed specifications A structured doc setting out detailed descriptions of the system services
Requirements Elicitation Techniques	-Individual Interviews -Group Meetings -Prototyping -Questionnaires -Observation -Research
Individual Interviews	2-way communication process Time sensitive Could be user, buyer, or expert
Group Interviews	N-way communication Groups of customers, cross-functional teams, buyers, experts, focus groups etc. Semi-structured Cons: Group think
Prototyping	A structure for individual or group exploration Participants are end users

	Cons: Could push yourself into a corner early on
Questionnaires	1-way communication Possibly anonymous Cons: False answers, Answer options that are too limiting, answer options that are too broad
Observation	Watch real people in the domain Ethnography Cons: Observing can cause behavior to change
Research	0-way communication Finding and reading written info and artifacts
Use Case	Describes sequences of events between an actor and a system that yield a result of value to the actor A template for a collection of related scenarios
Use Case Specification (Parts)	Objective Primary Actor Trigger Secondary Actor(s) Pre/Post Conditions Scenarios (Success/Failure)
Actor	Defines a coherent set of roles that users of an entity can play when interacting with the entity Stick figure

A use case should focus on the users _____	GOAL (you should avoid functional decomposition)
Steps to create a use case	1. Identify and Describe the Actors 2. Identify the Use Cases and write a brief description 3. Identify Actor to Use Case relationships 4. Outline the Individual Use Cases 5. Refine the Use Cases 6. Verify & Validate the Use Cases
<<include>>	A stereotype of a dependency A -> B The behavior of B is ALWAYS included into A
<<extend>>	A stereotype of a dependency A <- B A possible extension, behavior of B may be incorporated into A
Generalization	B inherits the behavior and communication relationships of A and is allowed to override and extend B is generally a standalone basic use case Actors may apply Generalization as well A <- B e.g. Student <- Graduate Student

Analysis Modeling Techniques	Data/object Models Behavioral Models Flow Models
Data/object Models	* Entity-Relationship (ER) OOA&D Data Dictionaries
Behavioral Models	Use Case Models State Machines
Flow Models	Process/workflow Models * Dataflow Diagrams (DFD) * Sequence Diagrams * Activity Diagrams

Requirements Quality Measures IEEE-830	Correct Unambiguous Complete Consistent Prioritized Verifiable Modifiable Traceable
Correctness	every requirement stated therein is one that the software shall meet
Unambiguous	If an only if it has only one interpretation
Complete	If and only if it describes all significant requirements of concern to the user Don't use etc Don't use TBD Hard to measure
Consistent	If an only if no subset of individual requirements described within it are in conflict with one another
Prioritized	Ranked by importance and stability
Verifiable	If an only if there exists a finite, cost-effective process with which a person or machine can determine that the developed software system does indeed meet the requirement Avoid writing in the negative
Modifiable	if an only if its structure and style are such that any changes to the requirements can be made easily, completely, and consistently, while retaining the existing structure and style of the set

Traceable	iff the origin of each of its component requirements is clear, and there is a mechanism that makes it feasible to refer to that requirement in future development efforts
Other Requirements Quality Measures	Clear Concise Cohesive Feasible Managed
IEEE 29148	A newer, longer doc (~100 pages). Focused on definitions
SWEBOK	Software Engineering Body of Knowledge Focused on descriptions
DO-178C	Avionics Software Standard Based on consequences of failure Full bidirectional traceability
Requirements Maturity Levels	0 - Chaos! No Requirements 1 - Written requirements 2 - Organized 3 - Structured 4 - Traced 5 - Integrated
Change Request Management	Single Channel for Approval Requirements to Design to Code to Test to Maintenance

Why is Traceability so important?	Quality -Can we determine that the req is validated/verified? Impact Analysis -What other reqs are impacted? -What people are affected? -What downstream artifacts are affected?
5 Step CM Process	1. Plan for change 2. Baseline the reqs 3. Change Control Board (CCB) 4. Use a Change Control System 5. Maintain Traceability
Types of Decomposition	Flow-down Refinement Completion
Flow-down Decomposition	- Assigning requirements to appropriate subsystems -An Architectural effort
Refinement Decomposition	-Ensure reqs reach level of specificity where implementation can easily follow -A Requirements effort
Completion Decomposition	-Adding reqs to complete missing back traces from code to reqs -Design or even implementation effort
Requirements specify ____ to build, not ____ to build it	what/how
SMT-LIB	well recognized standard for specifying formal constraints to be solved by an automated constraint solver