Module – II
Software Engineering Development Practices

Software Requirements

Table of Contents

Content Area – 1
Requirements Engineering Introduction

Software Engineering

- What is software engineering?
  - *Software Engineering is the application of engineering to software development*
- For a profession to be considered legitimate, it should satisfy three basic claims –
  - knowledge has to be validated by the community
  - validated knowledge is based on scientific ground
  - professional judgments and advice are oriented towards a set of substantive values

Software Engineering

Basic Pillars of an Engineering discipline

Computer Science | **Problem Solving** | **Discrete Mathematics**

Software engineering includes the knowledge, tools, and methods
- for defining software requirements
- performing software design, computer programming
- user interface design, software testing
- software maintenance tasks.

Additionally includes project management and quality management responsibilities
- Uses knowledge from many diverse fields like
  - computer engineering, computer science, mathematics
  - software ergonomics and systems engineering

Software Engineering - Software Life Cycle

A sample Software Life Cycle

Concept exploration – area where one would like to position the product

Requirements, **Design** and **Construction** part of the standard development phase

Testing, **Installation** and Check out part of the system validation process

Operation and **maintenance** involves mass deploying the product at customer sites and handling customer issues and entering the maintenance process

Retirement – phasing out the product and introducing a newer one in its place.

Software Engineering - Supporting Processes

Supporting Processes

Acquire – Initiation, RFP preparation, contract preparation and update

Supply – (Execution and Control) Initiation, preparation of response, contract, planning, define or use a software life cycle model, map processes

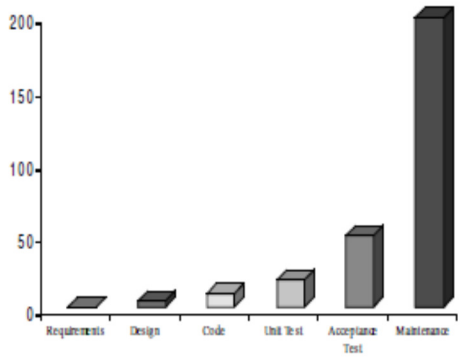
Develop - define or use a software life cycle model, map processes Software requirements, system architecture, analysis, design, test, integrate, …

Maintain – problem and maintenance analysis, modification implementation, migration and software retirement.

Definition of software requirement

- Requirements engineering is usually the first step in any software intensive development lifecycle irrespective of model
  - Usually difficult, error prone and costly
  - Critical for successful development of all down stream activities
  - Errors introduced during requirements phase if not handled properly will propagate into the subsequent phases
    - Resulting in a product that is not usable
    - Intended use will not be met

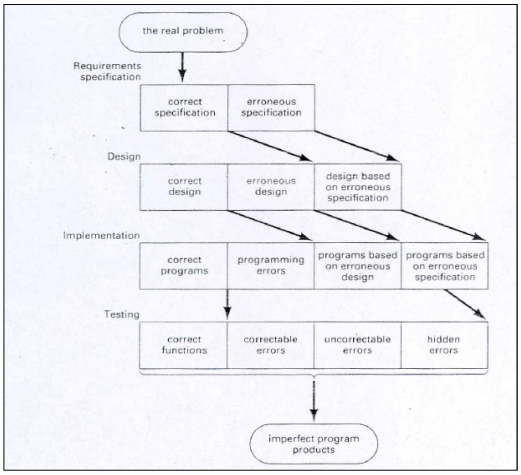
Cost of Repair



Life Cycle Stage	Relative cost of Repair
Requirements	0.1 to 0.2
Design	0.5
Coding	1
Unit test	2
Acceptance test	5
Maintenance	20

This can be found in any text book quantifying cost of repair in software engineering example Software Requirements Revision Objects, Functions, & States, by Alan M Davis Prentice Hall PTR, Englewood Cliffs, NJ, 1994

Propagation of errors



Reference : Mizuno Y Software Quality Improvement, IEEE Computer 15, 3, pp 66-72, 1983

Why is Requirements Engineering important?

- Invalid requirements can lead to a non useful system in spite of careful development and adhering to process
- Unnecessary requirements can make the system cumbersome
- Late requirements lead to schedule slippage eventually resulting in cost overrun
- Invalid requirements can lead extra steps to perform simple operations for the customer
- A UI Customer can actually become fatigued using the system

Reference: Software Requirements Revision Objects, Functions, & States by Alan M Davis Prentice Hall PTR, Englewood Cliffs, NJ, 1994

Benefits of producing robust requirements

- Providing a documented basis of what the software product is to do
- Reducing the effort required to produce the desired product
- Provide a basis for estimation of cost and schedule
- Provide a baseline of capabilities that must be tested or otherwise verified and validated
- Facilitating future evolution, adaptation and migration of software items

Software Engineering Vol-1, Development Process Ed Richard Thayer and Mark Christiansen, Third edition, IEEE Computer Society Press

What do Requirements define?

- What the system must do and how well the system must perform
  - Specify the functional and capability characteristics of the software
  - Establish the interfaces external to the software item
  - Set forth the qualification criteria for software testing
  - Provide safety and security specifications
  - Set forth human factors engineering specifications for the user interface
  - Establish the data definitions and database and other specifications for the software

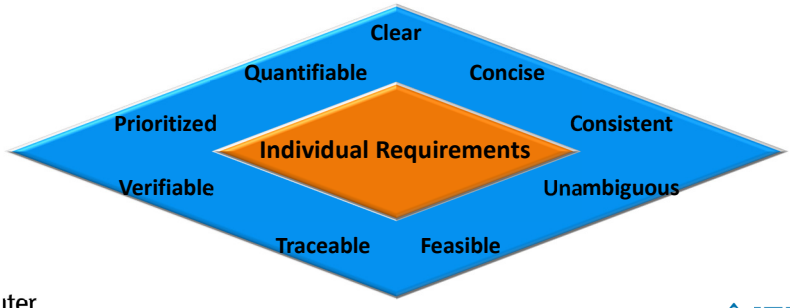
Software Requirements Definitions

- ***IEEE Std. 610.12-1990*** defines a requirement as:
  - (1) A condition or capability needed by a user to solve a problem or achieve an objective
  - (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents
  - (3) A documented representation of a condition or capability as in (1) or (2)
- The ***SWEBOK Guide 2004*** defines a software requirement as a "property which must be exhibited by software developed or adapted to solve a particular problem."

Software Requirements definition - SWEBOK

Properties of software requirements

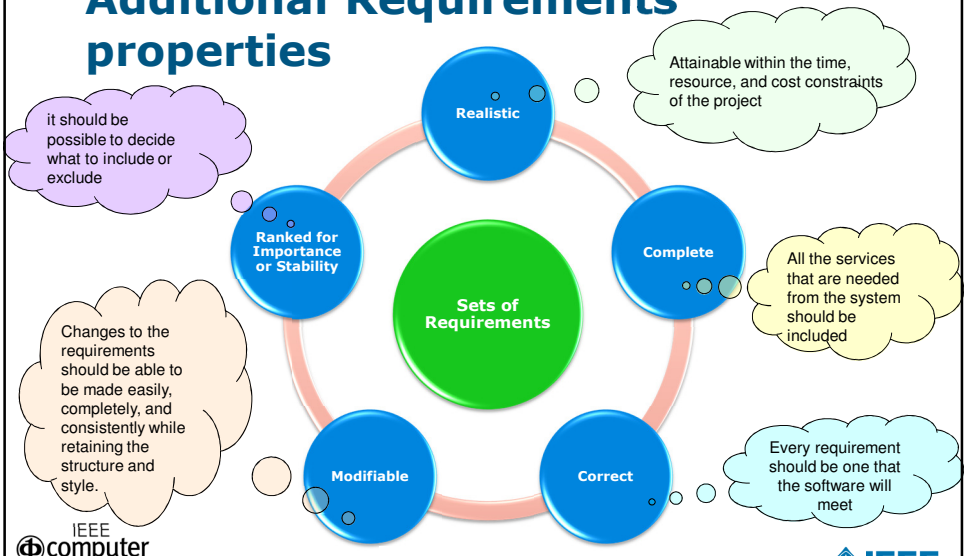
- Requirements must be considered from two aspects: that of individual requirements and that of the set of requirements as a whole.
- Individual requirements have the following properties:



Individual Requirement Properties

- **Clear** - Requirements should be written in precise, simple language that every reader can understand
- **Concise** – Requirements should describe a single property and expressed with as few words as possible
- **Consistent** - No requirement should contradict another
- **Unambiguous** - A requirement should have only one interpretation
- **Feasible** – realizable with a specified time frame
- **Traceable** – backwards to stakeholder request and forward to software components
- **Verifiable** - requirement must have a clear, testable criterion and a cost-effective process to check it has been realized as requested
- **Prioritized** - requirements should be prioritized
- **Quantifiable** - the requirement should be quantifiable, which aids in testing and verifying

Additional Requirements properties



Discussion Question

Which attribute of requirement is addressed here? Discuss this with respect to the statements in the previous slide

1. The system shall exhibit good response time
2. All customers shall have the same control field
3. The system shall be menu-driven
4. There shall be 25 buttons on the panel labeled PF1 to PF25
5. The software shall not exceed 128K of RAM
6. The elevator shall move smoothly and without jerking
7. The system shall be fail-safe
8. The phone shall ring.
9. Purchase order shall be raised when quantity reduces

Important points regarding requirements

- Requirements should specify all the externally visible behavior
- Requirements should describe the **WHAT** and not the **HOW**
  - **What and How** – Many a times the analyst gets lost into realizing the functionality and discussing the design approach for a requirement instead of really documenting what the software shall do
  - **What** the software shall do should be documented in detail during the requirements phase
    - Many a times requirements analysts get trapped into how to realize the feature than the define part
    - This usually results in a non scalable product
  - The **How** should be the topic of focus during design phase

Discussion Question

- *The system shall store the user names of all authorized users -*
  - in a database
  - In a flat file
  - some other form of persistent storage

Is this to be discussed and finalized during a requirements discussion phase? Why?

Product and Process Requirement

- **Product Requirements**
  - dictate what the software must do
  - what users must be able to do with the software
  - might give specific parameters for performance (memory, speed…)
  - portability (ability to move to a new platform)
  - usability (ease of use by the end user)

- **Process Requirements** are constraints on development of the software
  - development language or toolset
  - Verification techniques
  - overall process to be followed
  - imposed by users, the development organization, or third parties

Functional and Non-Functional Requirements

- **Functional Requirements**
  - Also called capabilities, state what the software must perform
  - The function can usually be directly observed, and, therefore, a test can be written to verify the function (behavior)
  - Verification of a well-written functional requirement is usually fairly easy

  Example :

  1. system shall assign a unique tracking number to each shipment
  2. system shall display the current payment due date as mmddyyyy

- **Non-Functional Requirements**
  - Specify the criteria that can be used to judge the operation of the system
  - Nonfunctional requirements may not be directly observable
  - Scope is the system as a whole and not individual components

  Example:

  1. With 100 concurrent users a database record shall be fetched over the network in less than 3ms.

Non-Functional Requirements

- **Non-Functional Requirements (NFRs)** – This is a criteria that is used to judge the overall operation of the system and not just individual behaviors
- These are extremely important from the overall system acceptance criteria by the customer since these represent the qualities of the system – either execution qualities of the system or evolution qualities
- Nonfunctional requirements apply to the system as a whole, not to individual features
- Some examples (typically an "…itility" is a NFR)
  - Maintainability, Usability, Portability, Scalability, Reliability, Safety, Understandability, Performance, ….

Some common NFRs … (1)

Availability	planned uptime, actually available and fully operational (MTBF)
Efficiency	how well the system uses the processor capacity, disk space, communication band width
Flexibility	how much effort is needed to add new capability to the product
Integrity	Precluding unauthorized access
Interoperability	exchange data or services with other systems
Reliability	software executing without failure
Robustness	degree to which the system continues to function, correctly when confronted with invalid data

Some common NFRs … (2)

Usability	ease to use and human engineering, user friendliness
Maintainability	how easy it is to correct a defect or make a change to the software
Portability	effort required to move or migrate software from one OS to an other
Reusability	extent to which a component designed for one application can be used in an totally different application
Testability	ease with which the software components or integrated products can be tested

Discussion Question

Why are the attributes in the table classified as important from the user and developer point of view? Discuss with examples.

Important primarily to users	Important to developers
1. Availability	1. Maintainability
2. Efficiency	2. Portability
3. Flexibility	3. Reusability
4. Integrity	4. Testability
5. Interoperability	
6. Reliability	
7. Robustness	
8. Usability	

Relationship between software quality attributes

	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efficiency			-		-	-	-	-		-	-	-
Flexibility		-		-		+	+	+			+	
Integrity		-			-				-		-	-
Interoperability		-	+	-			+					
Maintainability	+	-	+					+			+	
Portability		-	+		+	-			+		+	-
Reliability	+	-	+			+				+	+	+
Reusability		-	+	-	+	+	+	-			+	
Robustness	+	-						+				+
Testability	+	-	+			+		+				+
Usability		-								+	-	

Emergent Properties

- Many nonfunctional requirements are emergent properties
- Emergent properties are dependent on many factors, some of which are hard to analyze and control
- The system may have properties that relate to the system overall, not to individual components
- This is due to the complex interrelationships among components in a system
- In service-oriented applications, emergent properties are a significant portion of the requirements

Quantifiable Requirements

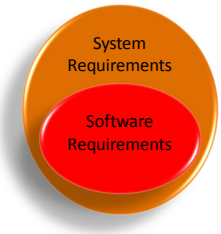
- Quantifiable requirements are measurable and verifiable
- Helps us to ensure that the requirements are clear and unambiguous

Quantifying Requirements

Requirement Type	Examples of Measures
Look and feel	Rate of acceptance
Usability	Error rates
Performance and speed	Response time
Reliability	Downtime
Portability	Number of platforms
Robustness	% of fatal/nonfatal errors
Maintainability	Time and work required to make a change
Size	Source lines of code (SLOC)
Certification	Compliance with standards

System and Software Requirements

***System* requirements** focus on the system as a whole	***Software* requirements** are those requirements that are derived from system requirements
They describe the externally visible behavior and operational constraints and answer the question, "What do you want the system to do? – Functional behavior Requirements may be allocated to hardware, software, organizational processes or other components of the system	These requirements are allocated to the software – What do we want the software in the system to do?



Discussion Question

Suppose a software company is developing an Automated Reservations system (AR) for an airline. Suppose the SRS contains the following statement:

1. The AR shall reduce airline labor costs.
2. AR shall be delivered within 10 months.
3. A passenger shall be able to use AR to make a flight reservation for a Fly-By-Night flight.
4. When a passenger submits credit card information, the AR will check name, credit card number, expiration date, and available credit.
5. An AR reference manual shall be developed.
6. A general user shall be able to use AR to check the status of a airline flight.
7. An AR user interface shall have a look and feel similar to that of a competitor's on-line reservation system.

Which of the above are functional and which are non-functional? Are there any problems with the above statements?

Content Area – 2
Requirements Engineering Process

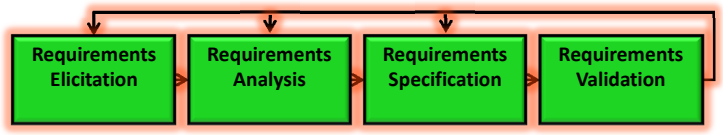
Process Models

- Process models are required to define the life cycle that would be used for the activity for example iterative, waterfall, etc.
- Requirements process models are a subset of the overall software development process model
- The requirements process takes a business or engineering problem and creates from it the specifications for a system that will provide a solution to that problem
- The process is never a strictly linear process

Iterative Process

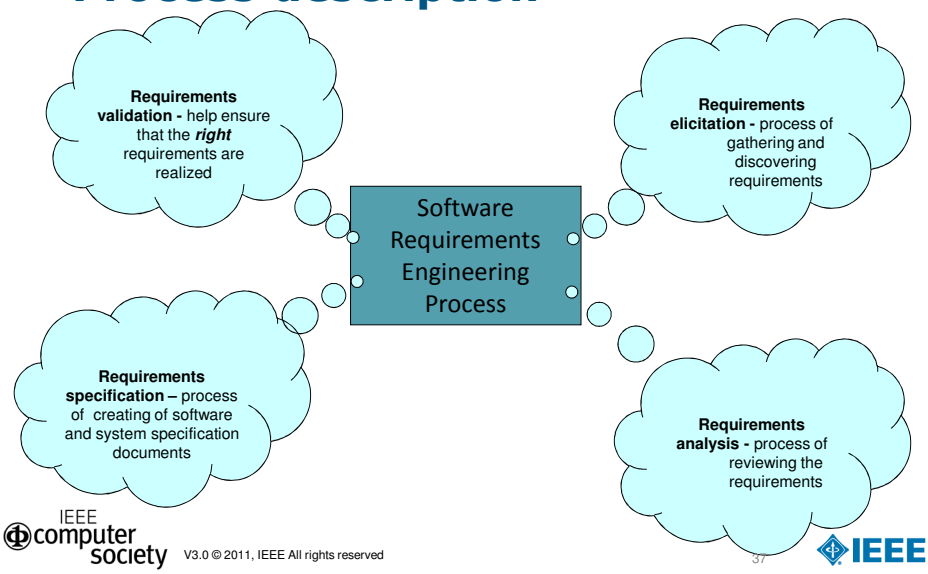
- There are four sets of activities that have been shown to produce specifications or requirements

Iterative Process



The project manager plans and executes this entire activity

Process description



Definition of Requirements Engineering Processes from SWEBOK - (1)

- **Requirements Elicitation** – whereby the information needed to develop and document the requirements are collected from the stake holders
- **Requirements Analysis** – which classifies the requirements, models the requirements, allocates them to a architectural design of the software and performs any tradeoffs and negotiations between the stakeholders

Definition of Requirements Engineering Processes from SWEBOK - (2)

- **Requirements Documentation** which creates the software and systems specifications needed to document the requirements in a manner suitable for use in further designing the software and in maintaining it
- **Requirements Validation** – which uses modeling, reviews, prototypes and ultimately acceptance testing of the final product to verify and validate the correctness of the requirements.

Aspects of Requirements Process

- This is not simply a front-end process
  - Requirements process is iterative irrespective of linear or incremental model selection
- Change is inevitable for requirements as we progress
  - Include provision for change in the planning process
  - All software requirements artifacts are configurable items and managed under a configuration management system, this will handle the change
- Requirements process needs to be flexible and easily tailored and adaptable to different organization contexts

Process Stakeholders

- These are people who have an interest in the software engineering project / product
- It is important to identify stakeholders early in the process
- This will help us ensure that all sources of requirements are included
- There can be different stakeholder groups like
  - **Users** who will operate the software.
  - **Customers** who have requested the software or who represent the target market.
  - **Market analysts** who determine the external market need and who may act as proxy customers.
  - **Regulators** which are representatives of third-party regulatory agencies.

Typical Process Stakeholders

Role	Use of Requirements
Project leader	▪ Determine scope and create project plan ▪ Get agreement from owners ▪ Track progress
Analyst	▪ Elicit requirements ▪ Decompose requirements
Development team	▪ Design and code to requirements ▪ Engineer efficiency and reuse
Test team	▪ Verify conformance to requirements
Legacy team	▪ Ensure legacy integration
Maintenance team	▪ Support users in production ▪ Ensure changes fit requirements
Project sponsor	▪ Provide motivation and sign off

Discussion Question

Suppose a software company is developing an Automated Reservations System for an airline. Who are the stakeholders?

Requirements Change Management

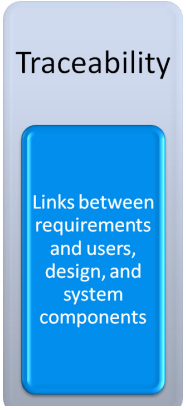
- Changes to requirements is inevitable particularly in large systems
  - It is difficult, if not impossible, to define or gather all the requirements at the very beginning of the project
  - As the problem becomes clearer, requirements will be either added, deleted or modified
- The requirements process provides support for these changes through a process called change management
  - Change control
  - Traceability
  - Version control
  - Status tracking

Requirements Change Control

- Change will impact work currently being done and work planned to be done
- Change during design and construction phase are more costly compared to changes in the requirements phase
- Requirements change process consists of
  - procedures for determining the impact of proposed requirement changes
  - management controls for responding to change requests

Requirements Traceability

- Traceability information allows the analyst to find links and assess the effect of a requirement change on other requirements and the work products
  - Many links exist between the requirements themselves and between the requirements and the users, the system design, and the system components



Requirements Version Control

- Purpose is to record all approved requirements change
- The changes have to be communicated to
  - Development team
  - Stakeholders
  - New specification document has to be written to reflect the changes and then released
  - There is numbering mechanism to identify the version

Requirements status tracking

- Status level will vary from project to project
- The status available for tracking are
  - Submitted
  - Pending
  - Reviewed / Terminated / Deferred / Rejected
  - Approved
  - Solved
  - Verified
  - Validated
  - Completed
  - Cancelled
- Each requirement must be uniquely identified to track status levels

Process quality and improvement

- One common measure of software quality is conformance to requirements
- Users review and agree to the requirements throughout the process
  - Ensures that defining the functionality and that decisions are not left for developers to make during development
- Cost to change a system at different stages will vary depending on
  - life-cycle model adopted
  - size and complexity of the system
  - Adherence to guidelines and standards

Content Area – 3
Requirements Elicitation

- *"The hardest single part of building a system is deciding what to build… No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later"*

  - Dr. Fredrick P Brooks Jr.

Requirements Elicitation

- Is a process of working proactively with all stakeholders gathering their needs, articulating their problem, identify and negotiate potential conflicts thereby establishing a clear scope and boundary for a project
- It can also be described as a process of ensuring that the stakeholders have been identified and they have been given an opportunity to explain their problem and needs and describe what they would like the new system to do

Dimensions of Requirements Elicitation

- Understanding the problem
- Understanding the domain
- Identifying clear business objectives
- Understanding the needs
- Understanding constraints of the system stake holders
- Writing business objectives for the project

Requirement Sources

Understanding the problem and domain

- Understand the context in which the software system will execute
- Break down domain into sub-domain areas
  - Done with the help of initial set of stake holders
- Subject matter experts are selected for each sub-domain area to explore each one fully
  - Getting right set of stake holders is a challenge
- Any accidental or deliberate omission of a stakeholder can have long term impact
  - Project boundary may not get well defined

Elicitation Techniques

- There are many different types of elicitation techniques
- The approach depends on
  - Nature of the system being developed
    - For example a UI intensive system needs a different approach compared to an embedded system
    - UI system the navigation and behavior is visible to all so easy to comprehend
  - Background and experience of stakeholders
    - If the stakeholder is a very hands-on person then prototyping and simulation would work
    - For non-computer savvy persons documents would be necessary

Classification of Elicitation techniques

- **Active** - where there are ongoing interaction between the stake holders and users. Some of the techniques are
  - interviews
  - facilitated meetings
  - role-playing
  - Prototypes
  - observation
  - scenarios
- **Passive** - infrequent interaction between the stake holders and users. E.g. use of
  - use cases
  - business process analysis and modeling
  - workflows
  - questionnaires
  - checklists
  - documentation
  - viewpoints

Active Elicitation Techniques - 1

- **Interviews** may be formal or informal, cost effective, specific open ended and close ended questions are used in a single session
  - interviews are not effective for getting requirements from the application domain, because either the language is so filled with business-related jargon that the engineers may misunderstand or domain knowledge
  - may not be mentioned by the stakeholder
- **Facilitated meetings** also called collaborative sessions, used when
  - stakeholders are dispersed or difficult to identify
  - trying to define the initial, high-level requirements at the beginning of a project
  - trying to identify and negotiate conflicts among requirements and stakeholders
  - facilitator needs to be neutral during the session

Active Elicitation Techniques - 2

- **Role-playing is a surrogate technique**
  - used when stakeholders are not readily available
  - product being developed will be mass-marketed
- **Prototypes** useful for eliciting an initial set of requirements quickly
  - Low-tech or low-fidelity prototypes
    - can be created quickly with pen, paper, post-it notes etc.
    - inexpensive to produce
    - users are very comfortable making suggestions
  - High-tech or high-fidelity prototypes
    - users interact with something that resembles the final product
    - new requirements can be produced based on new understand

Active Elicitation Techniques - 3

- **Observation -** end users allows the analyst to see what actually happens in practice
  - useful when users are unable to completely articulate their needs
  - analyst takes notes to capture flow sequences, scope for improvement, short cuts used, work around for power user
  - asks questions to discover why something is being done that way
  - observation often provides insights that clarify what needs to be built
- **Scenarios -** allow users to relate to real-life examples
  - scenarios may add detail to an outline or high-level requirements
  - typically start with a description of an interaction
  - provide a complete description of an interaction

Passive Elicitation Techniques - 1

- **Use cases** describe the interaction between a primary system actor and the system itself
  - contains sequence of simple steps
  - describes how to achieve a goal or task
  - large number of use cases are generally required to capture the complete flow of a system
- **Business process analysis and modeling -** analyst and the users analyze existing and future processes
  - process improvements identified by process analysis are the basis for system and software requirements

Passive Elicitation Techniques - 2

- **Workflows** are reliable repeatable patterns of activity designed to achieve processing intents of some sort
  - used to capture and develop the human-machine interaction
- **Questionnaires** are useful when a specific set of questions can be written
  - possible when the problem is well defined up front
  - used frequently in market surveys, general purpose products
  - users of existing systems can request new feature addition
- **Checklists** can be used to help stakeholders identify nonfunctional requirements (NFRs)
  - used for triggering discussion
  - NFRs are more difficult to identify and quantify early on
  - not taking NFRs into account are primary reasons for large system failures

Passive Elicitation Techniques - 3

- **Documentation** includes problem reports, designs, user manuals and manuals from competitors' products
  - can provide significant insight into possible requirements
- **Viewpoints** are useful
  - they recognize several different perspectives
  - provide a framework for discovering the conflicts in requirements
  - benefits of this approach include
    - Stakeholder buy-in and traceability
    - Flexible collaboration
    - Richer modeling structures

Discussion Question

Which of the following are active elicitation techniques and which are passive? And Why?

a) Use cases, workflows, and viewpoints
b) Observation, interviews, and facilitated meetings
c) Role-playing, prototypes, and scenarios
d) Business process analysis and modeling
e) Questionnaires, checklists, and documentation

Answer: Active: b, c; Passive: a, d, e

Discussion Question

What are the problems in requirements elicitation?

Issues and Problems in Requirements Elicitation - 1

- **Elicitation from stake holders may become a painful, drawn-out and thankless job**
  - elicitation should be carried out by senior and experienced staff with training in elicitation
  - focus should be on the WHY questions
- **Wrong stakeholders / failure to identify correct stakeholders**
  - may not speak for the entire community
- **Untrained Analysts in the assignment**
  - may not capture all the information provided accurately miss important ones
- **Incorrect levels of requirement levels**
  - stakeholders beating all over the place can result in chaotic meetings
- **Failure to collect enough information**

Issues and Problems in Requirements Elicitation - 2

- **System boundaries are not defined nor identified**
  - can result in cost over run, requirements creep-in, wrong expectations and wrong product delivery to customers
- **Understanding of product needs incomplete**
  - uncertain product needs by stake holders
  - uncertain of the business goals
- **Misunderstanding of computer's capabilities**
  - futuristic, wishful thinking, impractical expectations from stake holders leading to non realizable requirements
- **Communication gap between analysts and stakeholders**
  - they speak two different languages, analysts speak programming language where as stakeholders use technical jargon like medical terms
- **Stake holders don't communicate tacit knowledge or common well known information**
  - thinking that this is too obvious and every one is expected to know

Classroom Exercise – writing requirements

Divide into groups of two persons. Each person will receive a handout with some geometric shapes on it.

- Do not let your partner see your handout at any time.
- The only form of communication with your partner should be as specified below.

1. Write a text-only specification describing how to recreate the diagram on the page.
2. After 15 minutes, trade specifications with your partner and recreate your partner's diagram using only the specifications that he or she wrote.

Classroom Exercise – writing requirements

Discuss as a group how each of the following steps went and what could be improved:

1. Write a text-only specification describing how to create the diagram on the handout.
2. Recreate your partner's figure using only the text he or she wrote.

Content Area – 4
Requirements Analysis

Requirements Classification

- Requirements are broadly classified by
  - **Priority**
  - **Volatility**
  - **Source**
  - **Type**
  - **Risk**

Requirements Classification: Priority

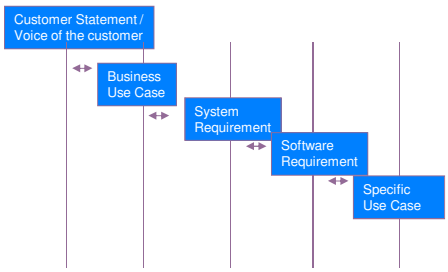
- **Must have -** Requirements that must be satisfied in the next release
- **Maybe -** Requirements that are desirable but need to be weighed in light of cost and other resource constraints
- **Not necessary (Optional) -** Requirements that can clearly be set aside for some later release

IEEE Std. 830-1998

- ***Essential*** - Implies that the software will not be acceptable unless these requirements are provided in an agreed manner
- ***Conditional*** - Implies that these are requirements that would enhance the software product, but would not make it unacceptable if they are absent
- ***Optional*** - Implies a class of functions that may or may not be worthwhile

Requirements Classification: Source and Volatility

- Source is related to traceability



- Degree of volatility is a measure of how much a requirement can change over time
- Sommerville defines the following
  - **Mutable:** changes due to the environment
  - **Emergent:** customer's understanding increases over time leading to changes
  - **Consequential**: changes due to new hardware/system
  - **Compatibility:** changes due to changes in processes

Requirements Classification: Type and Risk

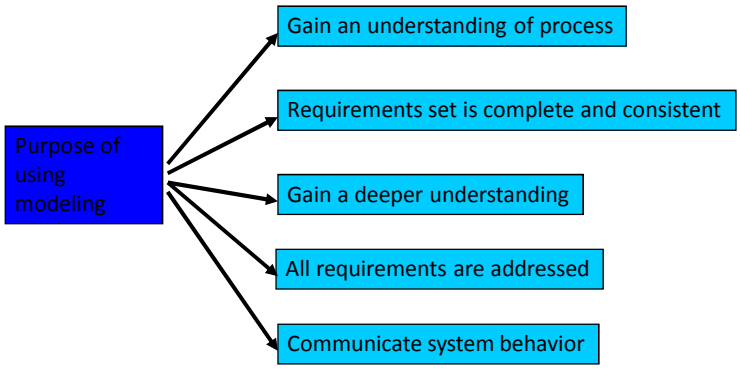
Type

- Functional
- Non-Functional
- Safety – hazard related
- Regulatory – conformance required to industry standards
- User Interface

Risk

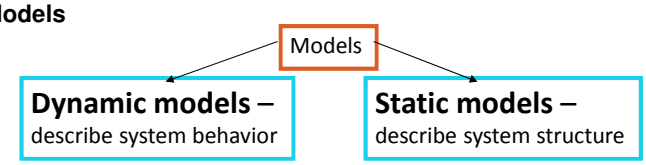
- If not addressed properly, can impact budget, schedule, quality and acceptance by end customer
- Risk can be caused due to
  - New / Complex requirement
  - New Technology
  - Organization integration

Conceptual Modeling



Modeling Types in RE

Types of Models



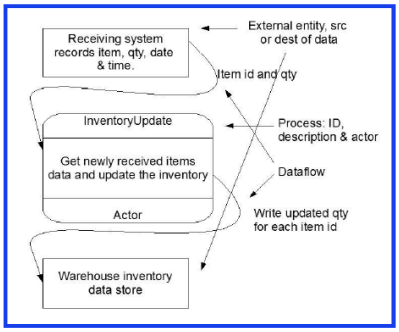
Modeling Notations

UML – used to depict use cases, state changes, and event sequences etc.

Integrated definition (IDEF) Languages
IDEF0 - functional modeling
IDEF1X - information modeling

Dynamic Model: Data-flow diagram

- Describe how data moves around a system in a sequence of processing steps
- **Valuable** because they can be used to show end-to-end processing
- **Intuitive** and easy to explain to an end user



Dynamic Model: Control Flow Diagram

- Similar to data flow models, but they show the control points of the system
- Control flows can show multiple paths through a system

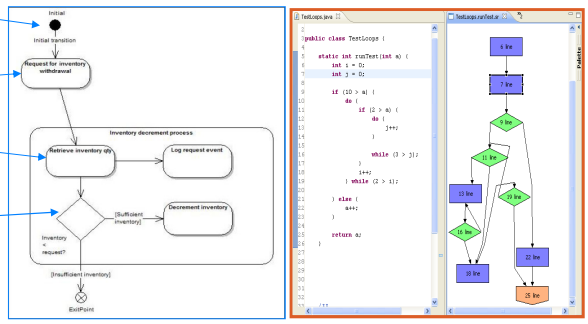
Defines a starting point, "Initial," and describes the activities that occur and their sequence

The process for decrementing inventory for an item

Withdrawal may come from a person at a terminal or another system

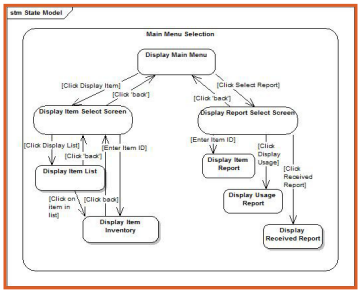
The inventory decrement process is started, and the warehouse system retrieves the quantity of the item and compares the available quantity

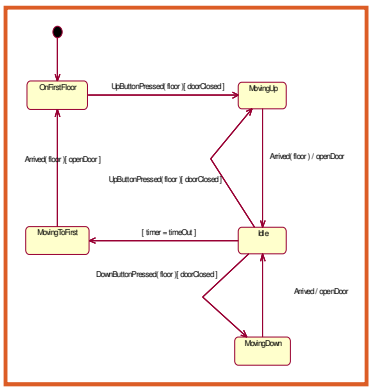
If there is insufficient quantity, the requirement is to deny the request



Dynamic Model: State machine models

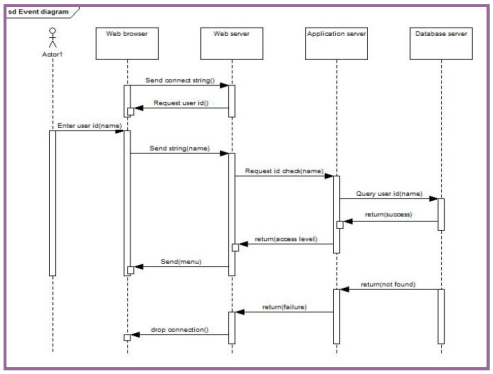
- The *state machine* is a model of behavior composed of a finite number of states, transitions between those states, and actions
- Example -





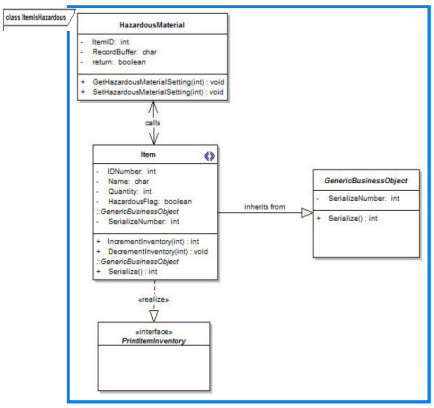
Dynamic Model: Event Tracing Diagrams

- Also known as sequence diagrams
- *Event tracing* - detailed information about how a process runs
- By including them during the requirements phase the customer gets a clear idea of the flow of events



Static Model: Class Models

- Using symbols and notations map requirements into OO language
- Models present familiar objects within their business so communication with end users easy
- Class contains data, information concerning a business entity, rules and processes governing its use



Content Area – 4 : Requirements Analysis – Static Models

Static Model: Personas – user interaction

- Personas, or user interactions, provide a way of thinking about and creating products for end users
- They provide insight into how users behave, how they think, what they wish to accomplish, and why
- Are based on patterns uncovered during interviews and observation of users

Barbara is a person who is likely to use a job search website. She is a 37-year-old single mother of twins. She is currently employed as a retail manager in a large department store chain. The company just announced that it will be merging with another store chain and will be laying off approximately 500 workers. She needs to find work quickly and does not want to relocate. Barbara has good Internet skills.

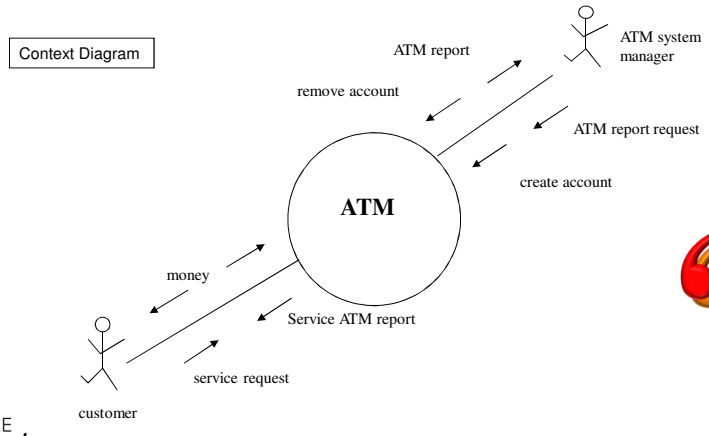
Sample Persona

Discussion Question

Develop an Automatic Teller Machine (ATM) banking system that will interact with banking customers to provide automated banking services (deposit money, withdraw money, provide account information - balance, transaction information, etc.). An ATM system manager can create a new account or close-out an existing account.

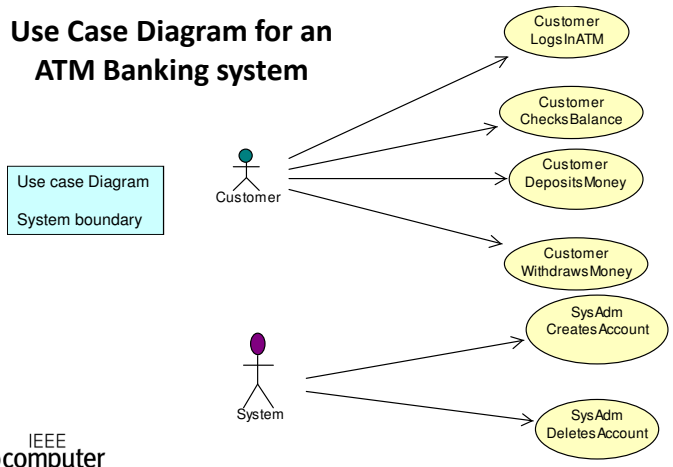
How would you carry out an **Object Oriented Analysis** of the ATM System?

Discussion Question: Solution - Context Diagram

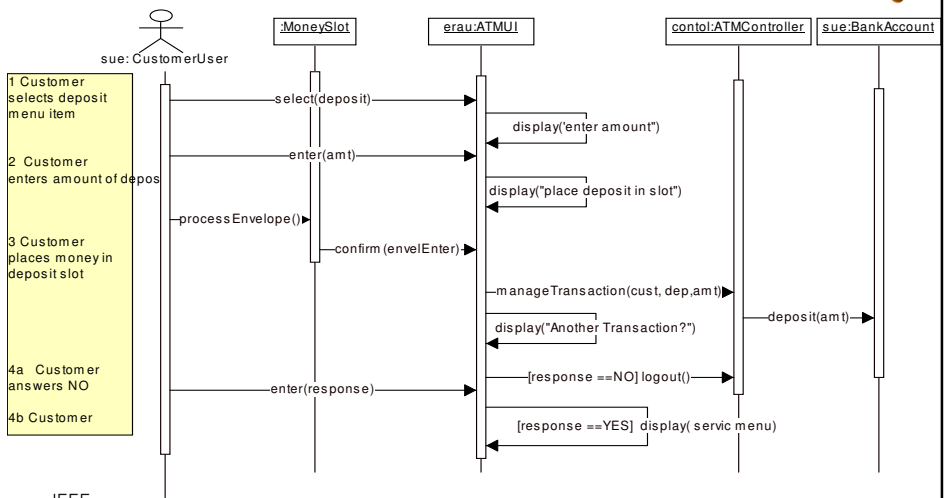


Discussion Question: Solution - Use Cases

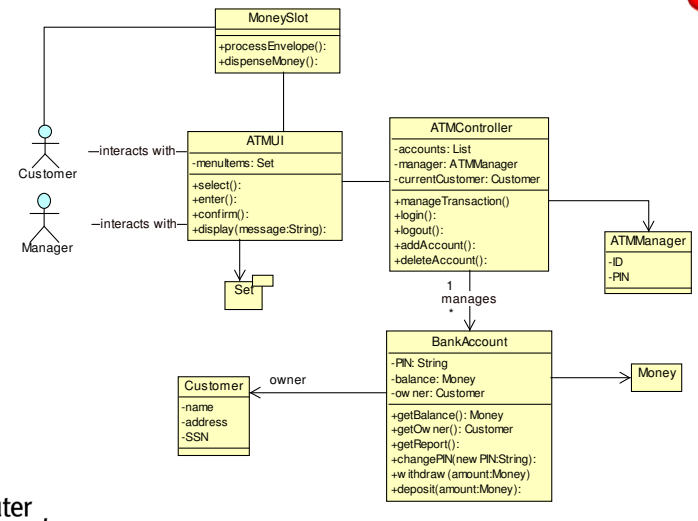
Use Case Diagram for an ATM Banking system



Discussion Question: Solution - Event tracing diagram - ATM

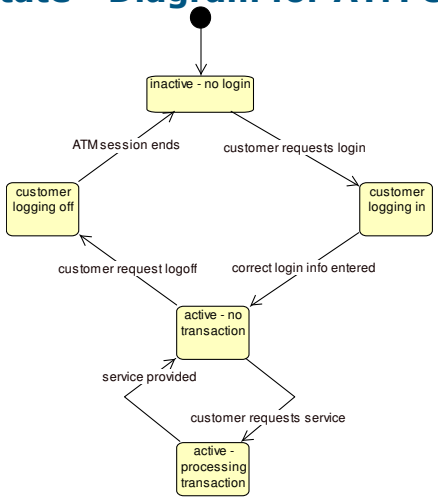


Discussion Question: Solution Class - Diagram ATM



Discussion Question Solution State - Diagram for ATM example

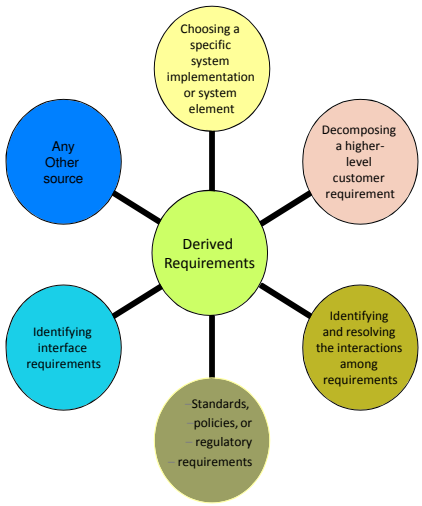
State Diagram for ATM User Interface



Derived Requirements

As user requirements are analyzed in more detail, the analyst may find other requirements that are based on or derived from those higher-level requirements

It is important to remember that derived requirements do not come directly from the customer or other high-level requirements source



Content Area – 4 : Requirements Analysis – Derived Requirements

Derived Requirements

Categories of Derived Requirements

Subsystem requirements are those that do not offer a direct benefit to the user but are necessary for the subsystems to operate properly

1. Customer may not have requested this
2. No business use case
3. module integration and system level test case will exist

Interface requirements appear when components need to talk to each other, pass information and / or share resources

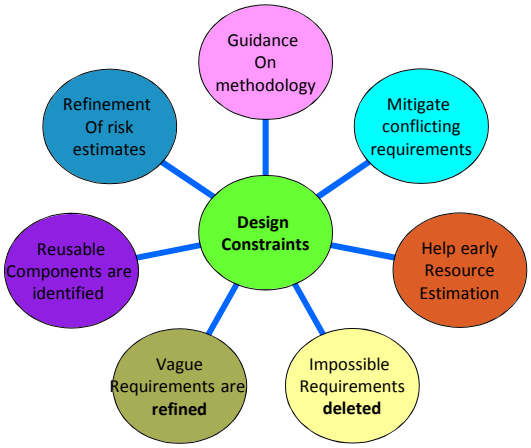
1. Cross dependency between components
2. Data send and receive
3. Access to a feature based on role
4. Licensed features

Software architecture, design and requirements allocation

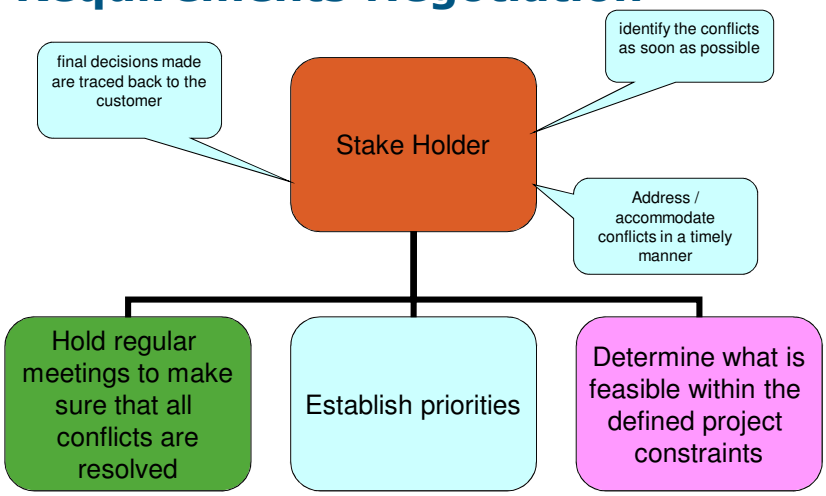
- During requirements analysis, it is appropriate to begin high level software architectural design even though requirements analysis and validation will continue
- Pay careful attention to nonfunctional requirements
  - can have far reaching effects on the system
  - may be allocated to many components
- The requirements analyst consult closely with the designer
  - to identify software components that satisfy the requirements
  - to capture new requirements as they emerge

Benefits of discussion between analyst and designer

Ensure that designers do not get attached to a solution they imagine even when there are no significant change to requirements



Requirements Negotiation



Discussion Question

Divide the class into five groups and assign one topic from the below list to each of them.

For the automated teller machine discuss

- the software architecture
- high level components
- user work flow
- nice to have features
- priorities and design constraints.

Discuss the findings and conclusions.

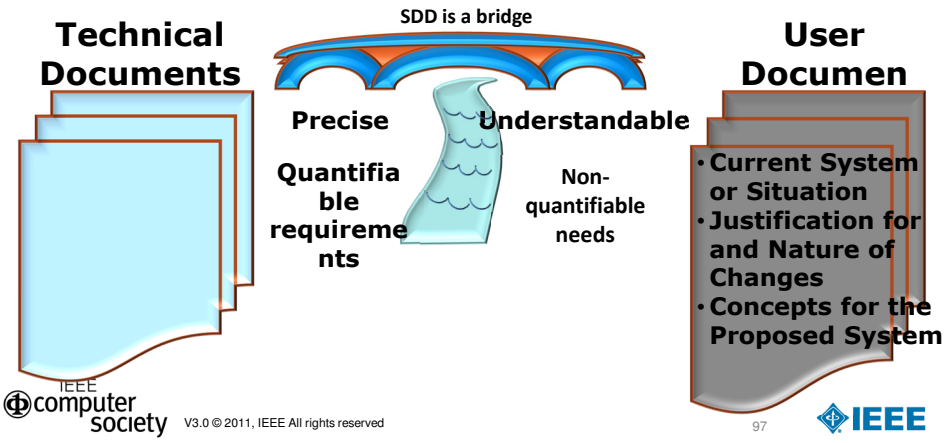
Content Area – 5
Requirements Specification

Requirements Specification

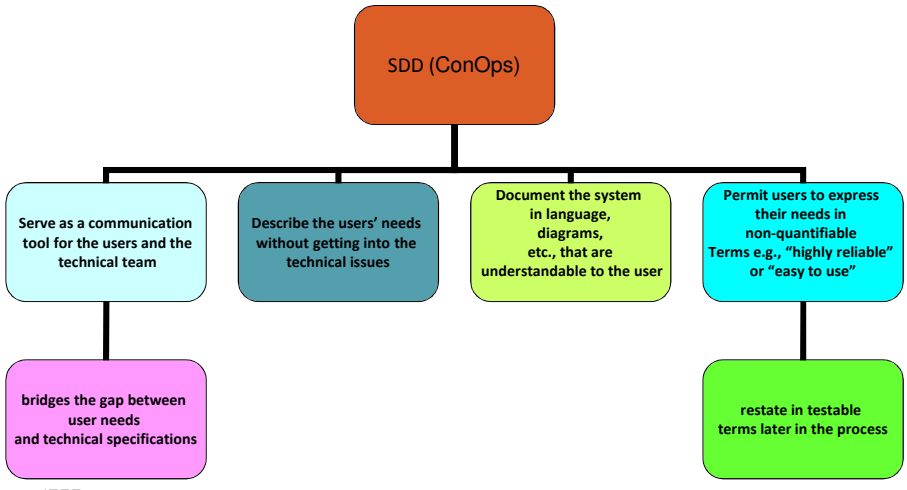
- Requirements specification is the documentation of a set of requirements that is reviewed and approved by the customer and provides direction for the software construction activities in the next stage of the life cycle
- For complex software systems
  - a layered requirements set is produced starting from high level system definition document
  - later lower-level specifications through analysis and decomposition producing a system specification is derived

System Definition Document

- Concept of operations (ConOps) document or SDD
  - User-oriented (user, buyer, developer, training)



Purpose of the SDD



IEEE Std. 1362-1998

- A ConOps is a user-oriented document that describes system characteristics for a proposed system from the users' viewpoint.
  - is used to communicate overall quantitative and qualitative system characteristics to the user, buyer, developer, and other organizational elements (for example, training, facilities, staffing, and maintenance)
  - Is used to describe the user organization's, mission's, and organizational objectives from an integrated systems point of view.

Recommended table-of-contents by IEEE for ConOps (*IEEE Std.1362-1998)*

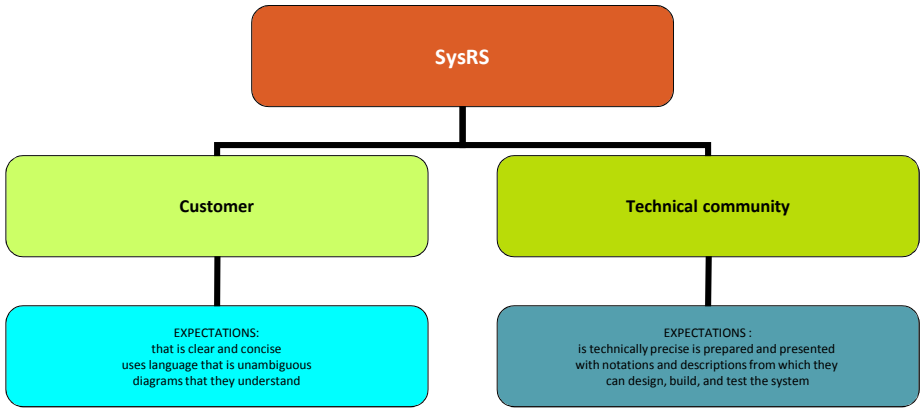
1. Scope
2. Referenced documents
3. Current system or situation
4. Justification for and nature of changes
5. Concepts for the proposed system
6. Operational scenarios
7. Summary of impacts
8. Analysis of the proposed system
9. Notes
10. Appendices
11. Glossary

Definition: System Requirements Specification - SysRS

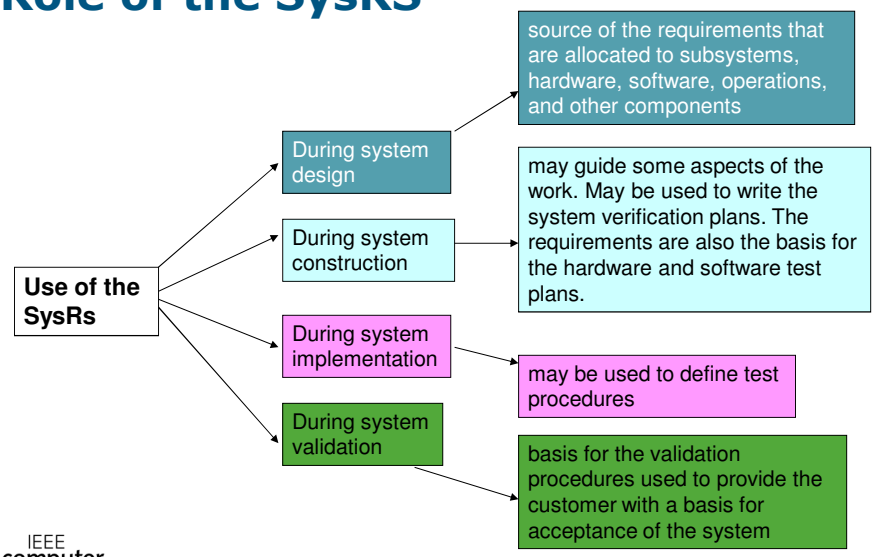
SysRS is the vehicle by which the customer requirements are communicated to the technical team that will design and build the system to meet the requirement.

This contains the subsystem allocation via subsystem requirements specification, hardware allocated, software allocated and manual process allocated.

Audience for System Requirements Specification - SysRS



Role of the SysRS



Discussion Question

A system requirements specification (SyRS) is a complete description of :

a) how a system should accomplish user requirements.

b) all inputs, outputs, and needed relationships between inputs and outputs.

c) tasks allocated to software and subsystems, but not to hardware or manual processes.

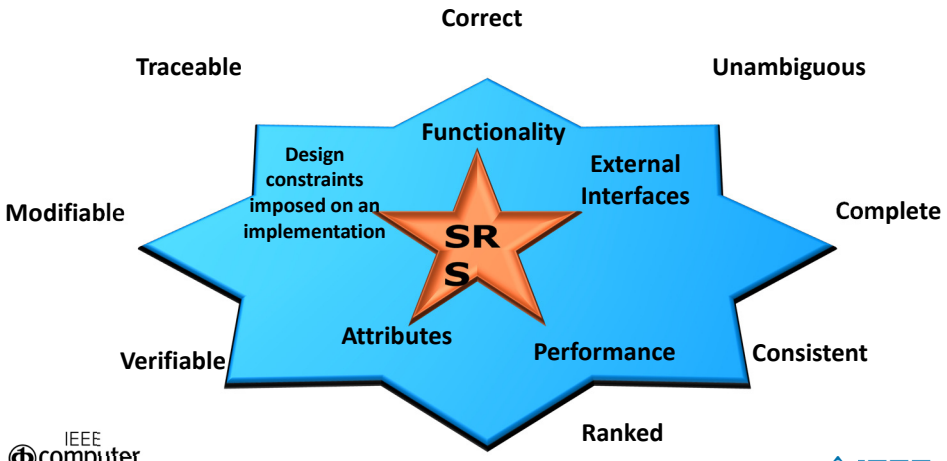
d) What a system should do, written for technical audiences only (rather than for customers).

Answer: b

Software Requirements Specification (SRS)

- The software requirements specification (SRS) document is the basis for
  - customers and contractors/suppliers agreeing on what the product will and will not do
- IEEE Std. 830-1998 on SRS - Indicates that "the SRS is a specification for a particular software product, program, or set of programs that performs certain functions in a specific environment."
- Describes both functional as well as non-functional requirements

Software Requirements Specification (SRS)



Software Requirements Specification (SRS)

Functionality. What is the software supposed to do?
External interfaces. How does the software interact with people, the system's hardware, other hardware, and other software?
Performance. What is the speed, availability, response time, recovery time, etc., of various software functions?
Attributes. What are the portability, correctness, maintainability, security, etc., considerations?
Design constraints imposed on an implementation

- required standards in effect
- implementation language
- policies for database integrity
- resource limits
- security
- operating environment(s), etc.?

Software Requirements Specification (SRS)

Correct - Every requirement in the SRS should be one that the software will meet
Unambiguous - Every requirement should have only one interpretation
Complete - Three elements must be present:
all significant requirements regardless of source or relationships
responses of the software to valid and invalid sources of data
full labels and references to all figures, tables, and diagrams
Consistent - The SRS must be consistent with other documents
Ranked for importance and/or stability
requirement should have an importance or stability indicator
Verifiable - Every requirement should be verifiable
Modifiable - Changes to requirements should be able to be made without affecting the structure and style of the SRS
Traceable - The origin of each requirement should be clear and the requirement should be able to be referenced in future stages

Table of contents recommended by IEEE for SRS (*IEEE Std. 830-1998)*

- 1. Introduction
  - 1.1 Purpose
  - 1.2 Scope
  - 1.3 Definitions, acronyms, and abbreviations
  - 1.4 References
  - 1.5 Overview
- 2. Overall description
  - 2.1 Product perspective
  - 2.2 Product functions
  - 2.3 User characteristics
  - 2.4 Constraints
  - 2.5 Assumptions and dependencies
- 3. Specific requirements
- Appendixes
- Index

Preparation of the SRS

- The following principles apply
  - Specific requirements should be correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable, and traceable
  - Specific requirements should be cross-referenced to other related documents
  - All requirements should be uniquely identifiable
  - Requirements should be organized to maximize readability
- May be written by the software engineer or analyst, or by representatives from the customer or both ensure that views and concerns from all parties are addressed

Content Area – 6
Requirements Validation

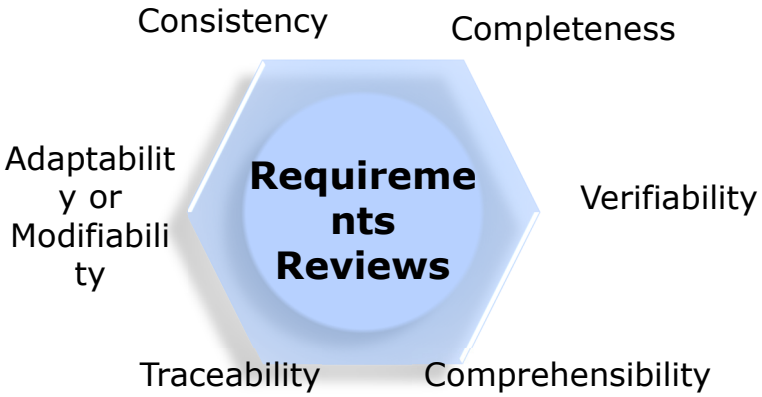
Requirements Validation

- The purpose of requirements validation is to help ensure that the requirements indeed does what customer wants
- This is an important phase because repairing requirements errors down stream can be expensive
- Verification and Validation
  - Validation determines whether the software requirements, if implemented, will solve the right problem and satisfy the intended user needs
  - Verification determines whether the requirements have been specified correctly

Requirements Review

- Requirements review is a common means of validation
  - Check the requirements documents for errors, omissions, conflicts, ambiguities and missing items
- Promote communication - potential problems can be detected early and resolved
- All requirements artifacts are configurable items
- Documents are distributed in advance so that the activity is effective and many defects are found early
- The results of the review should be documented and is under configuration management
  - include a list of identified problems and, if available, agreed-upon solutions
  - If no solutions were agreed upon, further meetings among stakeholders and developers may be needed

Requirements Reviews - checklist



Prototyping

- Prototype facilitate direct user involvement during requirements elicitation
- Often used with other forms of system modeling and simulations
- Ensures that the system engineer's interpretation of software requirements and those of the customer are the same
- Prototyping is most beneficial in systems
  - that will have many user interactions
  - where the human-interface interaction is a factor
    - Example - design of online billing systems where the use of screen dialogs is extensive
- Systems with little or no user interaction does not benefit from prototyping
  - such as batch processing or systems that mostly do calculations

Advantages and Disadvantages of Prototyping

Advantages

- Better quality and faster development at lower cost due to earlier discovery of correct requirements
- User involvement for better and more complete feedback
- Requirements are more stable when prototyping is used

Disadvantages

- Limited prototype can distract from big picture
- Look and feel could become focus over function
- Can be expensive
- May be throw-away code
- Using prototype as the production framework can be dangerous and costly

Model Validation

- All conceptual models must be validated at the end of the requirements activities to help ensure that all the essential requirements are present and the design is based on a true representation of the system

The objectives of model validation include:
- Ensuring that the models represent all essential functional requirements
- Demonstrating that each model is consistent in itself ensuring that, if multiple models were developed, the models are consistent with each other ensuring that the models represent the stakeholders' requirements

Example in flow diagrams, class models and static analysis verify that them proper communication paths exist to facilitate data exchange
For complex models use paraphrasing the model or translating it into a natural language description
If formal specifications are used, then formal reasoning may be applied used usually for high risk and hazardous systems Use of formal methods is expensive

Acceptance Criteria

- Requirements gathered from users and specified in the system requirements specification will be the basis for the acceptance tests
- Plan for Acceptance test during the requirements stage

Discussion Question

When should acceptance test criteria be determined? Why?

- During requirements elicitation?
- After the system architecture is determined?
- After unit testing and integration?

Content Area – 7
Practical Considerations

Dependency on Environment

- External business environment and processes change
  - creates a need for changes in the systems that support those processes
- Iterative Nature
- Handling Change

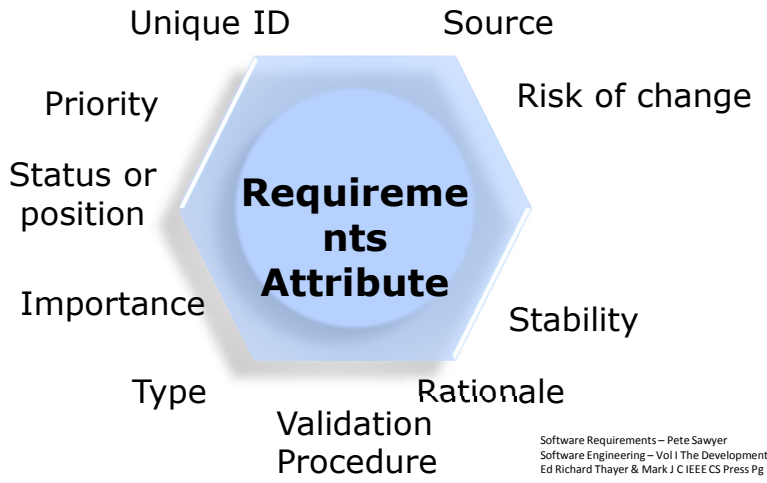
Iterative Nature	Handling Change
Requirements iterate toward a level sufficient for design Base-lining before all the requirements are fully understood can result in rework Project management plans may impose constraints on the software engineers Document all assumptions and known or potential problems	Understand system change through out the life cycle As the system is being developed there may be a need to change requirements or add new ones – change is part of the process – take steps to mitigate the effects of changes

Change Management

Iterative Nature

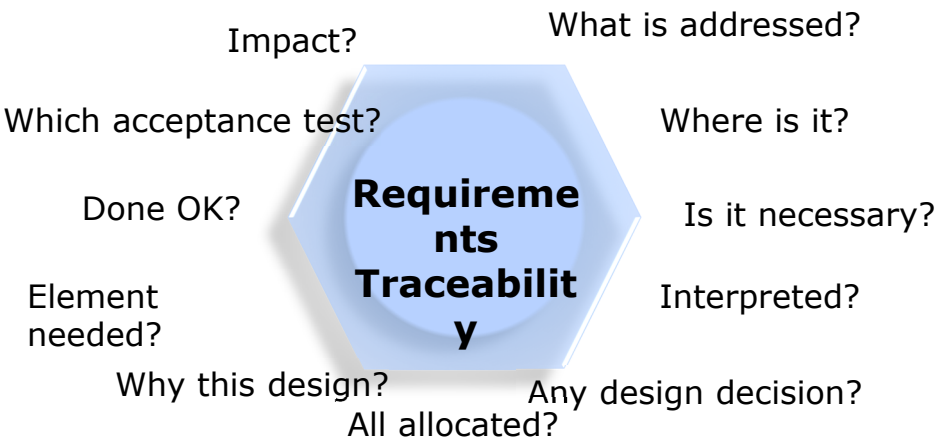
- Assess quality of requirements with requirements checklists
- Inform stakeholders of cost of change
- Establish change control procedures
- Use development approaches that accommodate change

Requirements Attributes

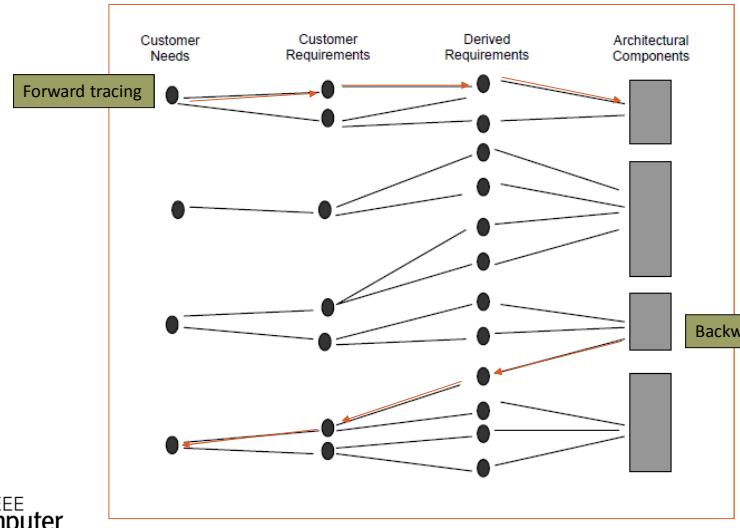


Software Requirements – Pete Sawyer
Software Engineering – Vol I The Development Process
Ed Richard Thayer & Mark J C IEEE CS Press Pg 133 (2005)

Requirements Tracing



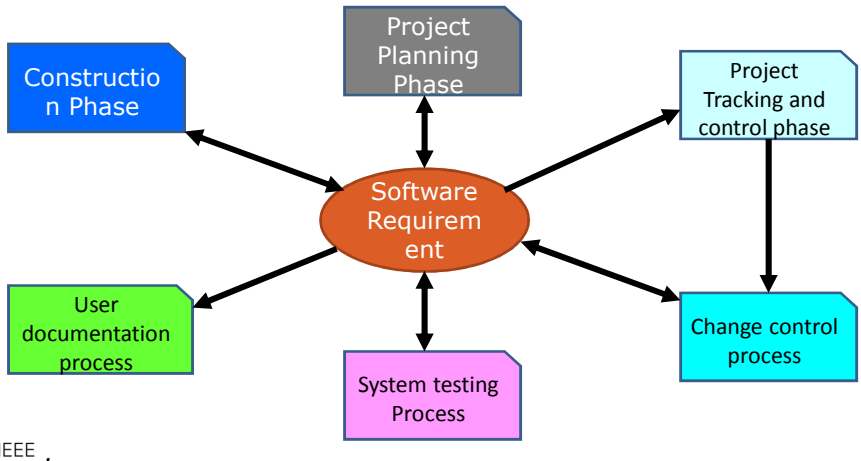
Requirements Traceability



Tracing tools and techniques

Tracing	Technique
Cross Referencing	Tagging, numbering or indexing requirements
Specialized templates	Store links between documents created during different phases of development
Restructuring	Use network or graphs to keep track of requirements change
Management tool	Automated tools like DOORS, Caliber-RM, Requisite Pro…

Relationship of requirements to other project process



Requirements Estimation Table

Measuring requirements

Content/ Service Area	High-Level Require-ments	Percentage Identified	Total High-Level Require-ments	Testable Requirements per High-Level Requirement	Total Requirements (High-Level × Testable)
Sales	20	80	25	5	125
Marketing	30	80	37	5	185
Student advising	20	70	28	10	280
Finance	30	100	30	5	150
Registrar	40	80	50	5	250

References and Additional Reading material

- Dorfman, Merlin. *Requirements Engineering,* reprinted from *Software Requirements Engineering,* 2nd ed., Richard H. Thayer and Merlin Dorfman, eds. Los Alamitos, California: IEEE Computer Society Press, 1977.
- *Guide to the Software Engineering Body of Knowledge (SWEBOK),* 2004 ed.Los Alamitos, California: IEEE Computer Society Press, 2004.
- Thayer, Richard H., and Mark J. Christensen, eds. *Software Engineering,Volume 1: The Development Process*, 3rd ed. Hoboken, New Jersey: John Wiley/Los Alamitos, California: IEEE Computer Society Press, 2005.
- Sommerville, Ian. *Software Engineering*, 8th ed. New York: Addison-Wesley, 2007
- Thayer, Richard H., and Merlin Dorfman, eds. *Software Engineering*, *Volume 2:The Supporting Processes*, 3rd ed. Hoboken, New Jersey: John Wiley/Los Alamitos, California: IEEE Computer Society Press, 2005