Requirements

"Conditions or capabilities needed by a user to solve a problem or achieve an objective"

"Conditions or capabilities that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document"

" A documented representation of conditions or capabilities as in the definitions above"

Getting the Requirements right is perhaps the most important part of the software development project!

Definitions from: IEEE Standard Glossary of Software Engineering Terminology

Requirements

- Understand Requirements
- Understand How Requirements are Gathered
  - Interviews
  - Brainstorming and Formal Sessions
  - Prototyping
- Understand Requirements Documentation
  - Scenarios
  - Use Cases
  - Models
  - Requirements Specifications
- Building Quality INTO Requirements
- Requirements Review and Acceptance

Developing Quality Requirements

- **What makes a 'Good' Software Requirement**
  - **Correct**
  - **Feasible**
  - **Necessary**
  - **Prioritized**
  - **Clear**
  - **Concise**
  - **Verifiable**

Requirements and Project Strategy

- Vision Statement
  - Provides all project participants a common understanding
- High level Business Goals & Objectives
  - Stated in user/customer terms
- Requirements must align with and enable the achievement of business strategies
- Derive Project Strategy from
  - Strategic Plan/Business Plan
  - Business Case/Feasibility Study
  - Project Charter/Project Plan
- Development of measurable goals & objectives helps the team prioritize requirements and resolve issues

Types of Requirements

- **User requirements – specify the user expectations in terms of the processes/tasks that the software will support**
- **Functional requirements – specify what the software has to do. May be called product features**
- **Non-functional requirements – mostly quality requirements that stipulate how well the software does what it has to do**
  - **Spoken and Unspoken requirements**
  - **Constraints**
  - **Architectural (i.e. interfaces, upgrades)**

User Requirements

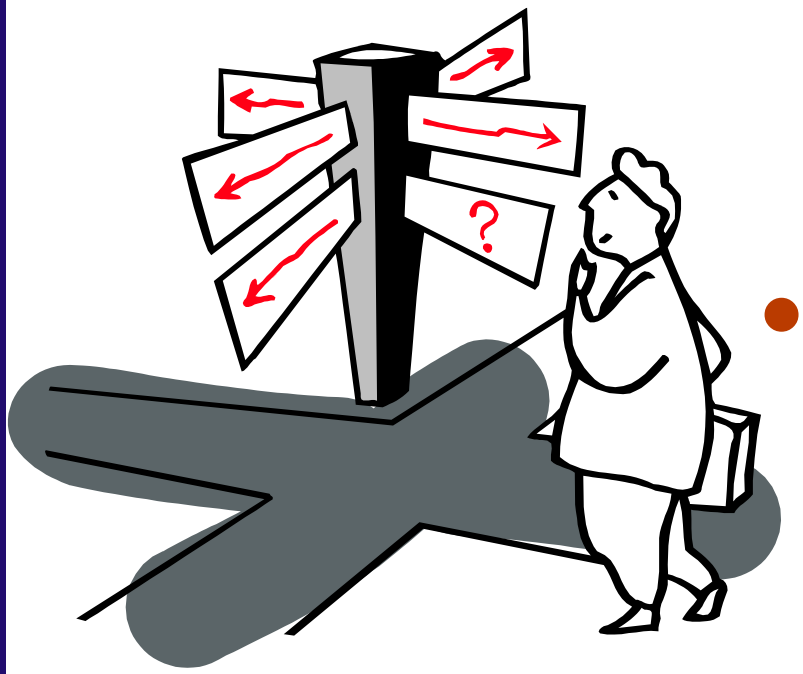
Voice of the User!

- **Defined by:**
  - **Customer**
  - **Marketing**
  - **Users**
- **Documented in:**
  - **User scenarios – unstructured description of user requirements**
  - **Use cases - a structured outline of interactions between user and system**
  - **Workflows**
  - **Prototypes**
- **Traceable back to Goals and Objectives**

User Groups

- Identify User Types / Classes by
  - Business function
  - Authority structure
- Aspects of their job tasks that might influence design
  - Skill levels
  - Frequency of use
  - Type of use

Product Champions



- Sets direction
  - Represents a user class
  - Serves as primary interface between users and developers;
- Must be:
  - Actual users
  - Not surrogate users such as funding sponsors or marketing staff
- Collect requirements from members of their user class
- Resolve differences of opinion

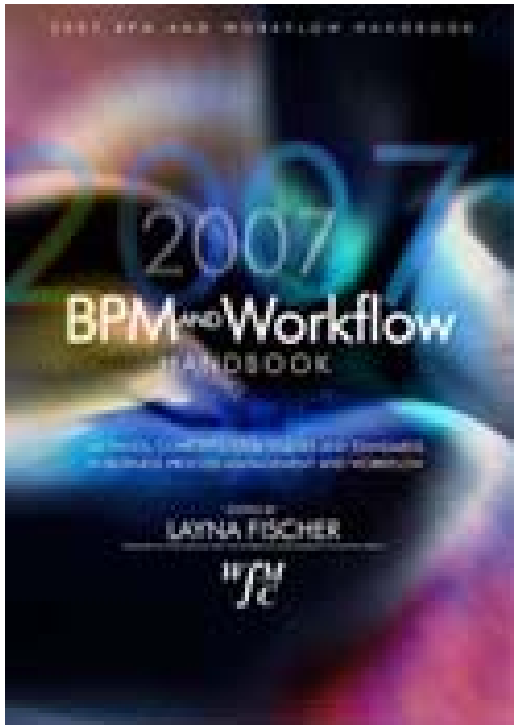
User Focus Groups

- Small groups of users
  - From previous product releases
  - Users of similar products
- Purpose: collect their job experiences (good and bad) including business and workflow knowledge
- Do not have decision making authority

Business Documentation

Business Process Management and Workflows

- **Defines the flow of a business process in response to a business event**
- **Includes roles, resources, inputs and outputs, decision points, hand-offs**
- **Identifies manual and automated activities**
- **May be used to reengineer or redesign business processes by identifying redundant, unnecessary, ineffective, inefficient activity**



http://www.e-workflow.org/

User Requirements Documentation

User Scenarios

- An unstructured narrative, a story
- Describes interactions between user and software
- Includes business goals and user motivations
- Describes the way the software is used in the context of doing business
- Contains minimal technical details
- May be used as justification – highlighting how the work process is enhanced by the software
- Often the basis for developing more detailed requirements

User Requirements Documentation

Use Cases

- Structured outline that describes the user's (actor's) interaction with the software
- Describes the course of events that will be carried out by the system
- A static description – meaning one actor, one initiating event, and the interaction that occurs at one time

Functional Requirements

- The functions that the system must provide to satisfy the requests by the user.
- Derived from the Use Requirements
- Usually done by the developers in collaboration with users/product champion
- Documented either:
  - Within the Use Case
  - Outside the Use Case

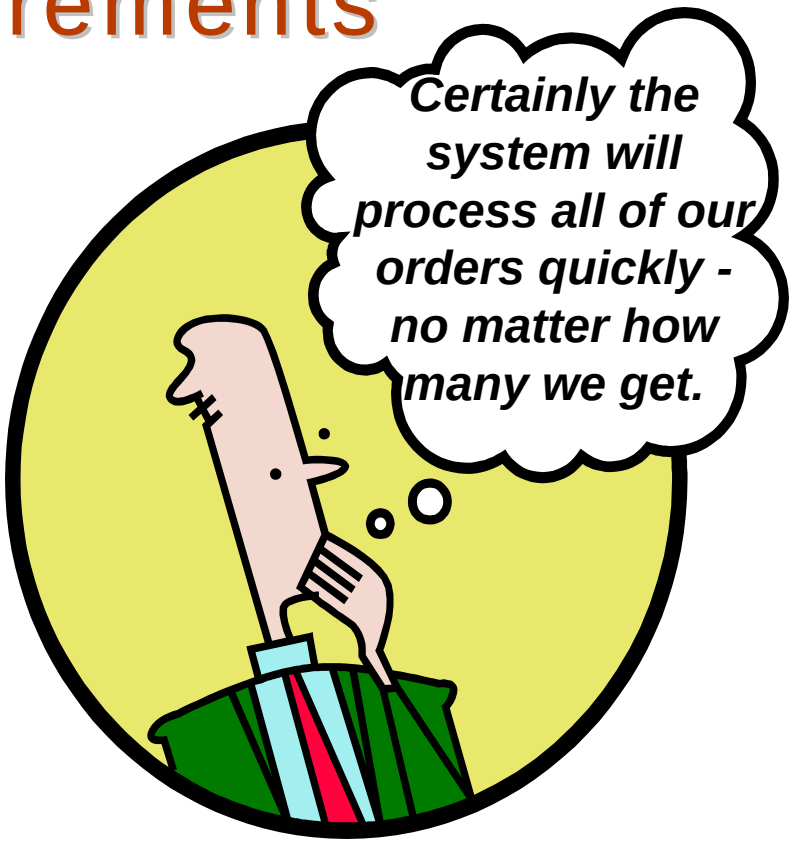
Functional Requirements

- **The Use Case without functional requirements is not enough detail for development of the software**

- **Functional requirements addresses the internal processing to be performed by the software**
  - **calculations**
  - **technical details**
  - **data manipulation**
  - **other specific functionality**

Non-functional Requirements

- **Often include implicit expectations about how well the product will work**
- **Impose constraints on the design or implementation (e.g. performance, security, standards, or design)**
- **May be general or global type requirements**



What are some examples?

Non-functional Requirements

- **Important non-functional requirements to users:**
  - **Availability**
  - **Flexibility**
  - **Reliability**
  - **Usability**
- **Important non-functional requirements to developers:**
  - **Maintainability**
  - **Portability**
  - **Testability**

Use Case Template

<Use Case ID> <Use Case Name>

<Created By> <Last Updated By>

<Date Created> <Date Last Updated>

Actors

<List all actors that will interact with this use case.>

Description (Goal)

<The goal of the use case describes what the use case will accomplish>

Pre-conditions

<List all situations that must be present prior to the initiation of this use case>

Use Case Template (continued)

Basic Course

1. <List all steps in the normal or *happy* path of the use case between the actor and the system>
   a. Actor Action:
      i. Identify what the actor will do to make a request of the system.
   b. System Response:
      i. Identify how the system must respond to the actor's request.

2. <Include any other use cases using the *Include:* construct>

3. <Extend this use case by branching off to other use cases using the *extends*: construct. This can also be documented using the Alternate Course section below>

Post-Conditions

<List the conditions that will be present upon successful completion of the use case>

Use Case Template (continued)

Alternate courses

<Another course that also results in a successful task completion of this use case by represents variations in the specifics of the task>

Alternate Post-Conditions

<As a result of an alternate course there may be alternate post conditions>

Issues and questions

<Used to track specific issues and questions concerning this use case only>

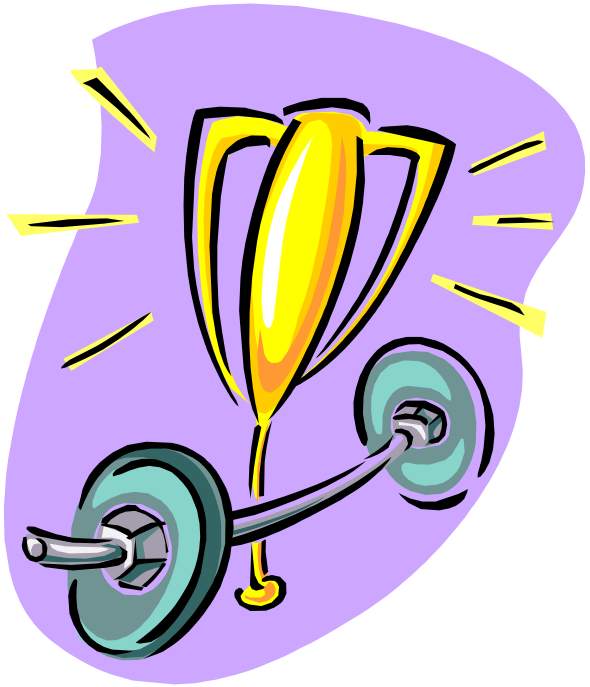
Priority

<Specify the priority of this use case amongst other use cases>

Frequency of Use

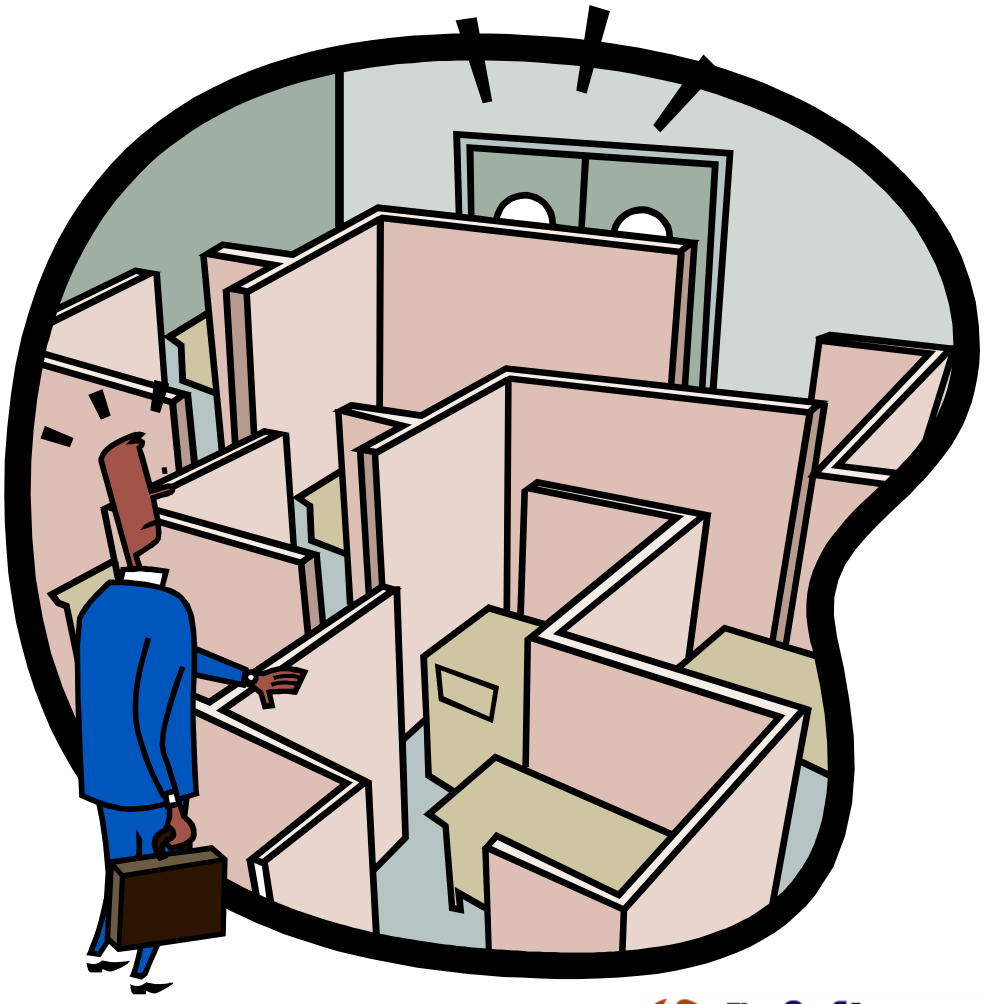
<Specify how often this use case is initiated by the actor(s)>

Use Case Benefits



- Consistency in documentation
- "Task Centric" perspective
- Actor - System Dialogue
- Use Cases easily transformed into Design
- Function can always be traced back to Use Cases
  - Modifications are easily identified
  - Reduces creation of orphan function

Use Case Traps to Avoid

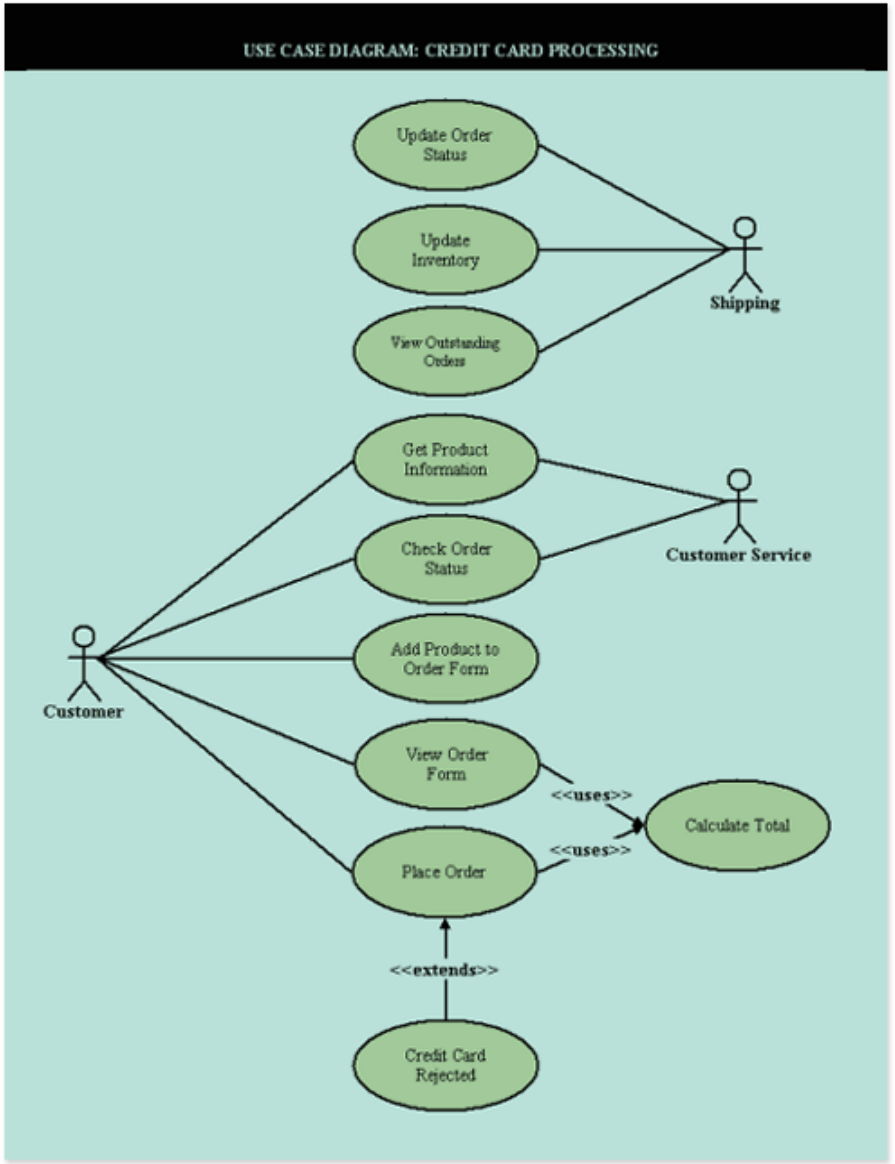


Avoid Use Case Explosion!

- Assign project champions (who are actually users/actors) to help develop use cases
- Avoid creating too many use cases to quickly
- Each use case tells a story about how the user interacts with the software to satisfy a discreet goal
- Avoid duplication across use cases
- There should be more functional requirements than use cases

Avoid Duplication Across Use Cases

- **Duplication is easiest to see by using a Use Case Diagram**
- **A use case diagram is a graphical depiction of use cases**
- **Shows the actors that initiate use cases**
- **Shows the interaction between use cases**



Avoid Design in Use Cases

- Discussing UI design is helpful in clarifying the requirements BUT should not be in the use case
  - Use case is a conceptual dialog between the actor and system only
  - Describe WHAT the system should do – NOT HOW the system does it
  - No tech language (i.e. windows, screens, fields, menus, data definitions, etc.)
- Specifying design features constrains and limits the design
- Prototypes are developed to help clarify requirements – but are loose representations of the design – a way to explore options

Use Case Example

UC - 7 Enter Written Request

Created by: k. cos 4/20/06 Last Update: k. cos 4/20/06

Actors

Public Information Officer
Authorized staff

Description (Goal)

Enter the details of a written open records request that has been received by mail or in person for tracking purposes. A copy of the original document will be retained as part of the electronic request.

Pre-conditions

The request does not already exist in the system.

Use Case Example

Basic Course

1. Actor Action:
   a. Enter the date the request was received
   b. Enter requester name, address and other available contact information
   c. Enter a description of the request
   d. Scan the written request document
2. System Response
   a. Save the request
   b. Link the scanned document to the request
   c. Generate 1st response due date (response date is 10 business days from date the date the request was received).
   d. Assign status – unassigned
3. The use case may be extended by UC-8 Assign Use Case.

Post-Conditions

The request must include:
- date the request was received
- minimum requester information (name and address)
- description of request
- scanned copy of the written request
- first response due date
- Status: Unassigned

Use Case Example

Alternate courses

If incomplete the Actor has the option to save a draft request

Alternate Post-Conditions

Save Draft request with information that has been entered
Status: Draft

Issues and questions

How long will a draft be retained by the system?

Priority

High

Frequency of Use

On average, 10 written requests per week

How can we gather requirements?

- **Brainstorming Sessions**
- **Joint Interactive Sessions (JRP, JAD, FAST)**
- **User Scenario and Use Case Development Sessions**

Brainstorming Sessions

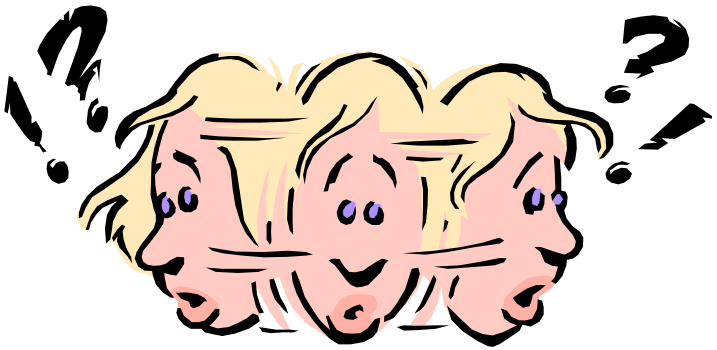
Provides a positive approach to developing ideas

- Practical Rules
  - Provide guidance to participants focusing on a given issue
  - Facilitate the session
  - Allow all participants to speak if they choose
  - No explanations only ideas
- Great way to lead into Joint sessions and Use Case development sessions

Joint Interactive Workshops

- Workshops are a people process
- JAD was developed by IBM in the late 70's
- Critical Success Factors
  - Requires a strong facilitator, unbiased and has no political stake in the outcome of the workshop
  - Requires a highly structured agenda with key goals
  - Requires an issues management process to resolve conflicts that arise in a timely manner
- Can be used to develop Use Cases

Are we done yet? How do you know?

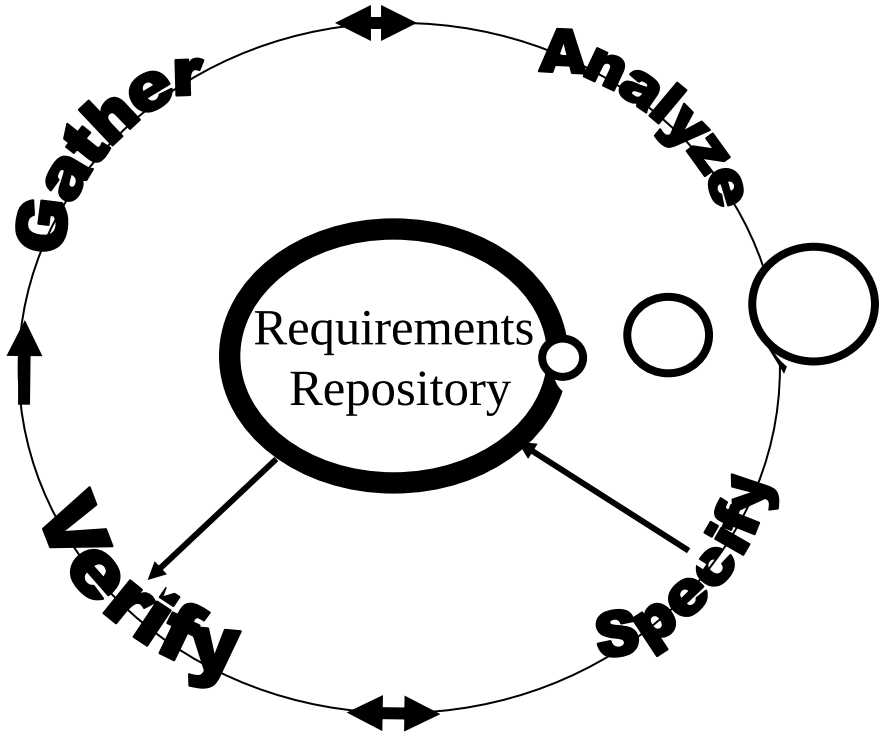


- During joint sessions users cannot identify any new Use Cases
- Proposed Use Cases are out of scope
- All business events are satisfied
- All goals and objectives are satisfied
- All business 'objects' are created and used

Post Workshop Activities

- Issues Management
- Finalize Workshop Documentation
- Develop Requirements Specifications
  - Request for proposal
  - Software acquisition
  - Prototyping
  - Software Requirements Specification
- If a prototype is developed then organize user workshops to review and validate the prototype

Requirements Repository



Use Cases;
User Scenarios
Workflows
Functional Reqts
Non-functional Reqts

Software Requirements Specification (SRS)

- **The SRS is a set of documentation describing what the software or system should provide to the end user in terms of function.**
- **Specifies each requirement discretely**
- **The document can include:**
  - **Text**
  - **Scenarios and Use Cases**
  - **Models or diagrams**

Developing the SRS

- Compile requirements
  - User requirements
  - Functional requirements
  - Non-functional requirements
- Evaluate and review requirements for quality
- Groundwork for Testing and Acceptance
  - Test Planning and Estimates
  - Test Strategy
  - Test Case Development

Benefits of the SRS

- Complete description of software functionality
  - Accurately describes user expectations
  - Provides developers with a clear understanding of exactly what the customer wants
- Baseline agreement between the users and providers of the software product
- Rigorous development of requirements with the customer reduces redesign, recoding and retesting
- Used as a basis for cost and schedule estimates
- Testing plans can be developed in a productive manner
- Easier transition of the product to users
- Basis for enhancements

Requirements Document

- **Typical Contents**
  - **Section 1 - Introduction**
    - **Purpose**
    - **Scope**
    - **Assumptions and Dependencies**
  - **Section 2 - General Description**
    - **Product Perspective**
    - **Product Functions**
    - **User Characteristics**
    - **Constraints**

Requirements Document (continued)

- **Section 3 - Specific Requirements**
  - **Functional Requirements–by function**
  - **Interface Requirements**
    - **User Interfaces**
    - **Hardware Interfaces**
    - **Software Interfaces**
    - **Communication Interfaces**
  - **Performance Requirements**
  - **Design Constraints**
  - **Quality Characteristics**
  - **Other Requirements**
    - **Database**
    - **Operations**
    - **Site Adaptation**

Requirements Document (continued)

- **Section 4 - Supporting Information**
  - **Definitions, Acronyms and Abbreviations**
  - **References**
  - **Appendices**
    - **Traceability Matrix**
    - **Models**
    - **Other information or documentation related to requirements**

Attributes of Quality Requirements Specification

- Correct
- Unambiguous
- Complete
- Consistent
- Verifiable
- Changeable
- Necessary
- Traceable

Correctness

- Requirements should accurately reflect the business needs as specified in
  - User scenarios
  - Use cases
  - Workflows
  - Business Case
- A traceability matrix is useful to determine correctness

Unambiguous

- Every requirement should have only one interpretation
- Define terms that may have multiple meanings in the glossary
- Refer to models and analysis documents as necessary

Completeness

- All significant requirements are included – more than functionality
- The response to all business events (inputs) have been defined for all situations
- All figures and diagrams have been referenced and accurately labeled
- No TBDs

Consistency

Requirements and subsets of requirements should not conflict with other requirements or subsets

- **Conflict in characteristics**
  - **One requirement says all alerts will be red, another requirement specifies that a specific alert is green**
- **Logical or temporal conflict**
  - **One requirement states two inputs shall be added, another requirement says the two inputs shall be multiplied**
  - **One requirement says A follows B, another says A and B occur together**
- **Same object, different terms**
  - **One requirement refers to an invoice, another refers to a statement.**

Verifiability

- A requirement must be finite and can be checked by a human or machine
- Watch out for ambiguous language
  - Works well
  - Shall usually happen
  - Good
- Some statements cannot be verified:
  - "The program shall never enter an infinite loop"
  - Testing for this is theoretically impossible
- Use of concrete terms and measurable quantities

Changeability

- Well-organized set of requirements accommodates changes while retaining structure and style
- Modifications can be made accurately and completely
- Avoid redundancy – requirements that appear in multiple places are more difficult to change
- Avoid circular specifications and interdependencies

Necessity

- Rank requirements for importance
- Rankings:
  - Essential – software is unacceptable without this requirement
  - Conditional –requirement would enhance the business function but the software would not be acceptable without it
  - Optional – the requirement is considered an potential opportunity

Traceability

- **Tracing requirements to their origin and through to implementation ensure fulfillment of the software product**
  - **Backward traceability – referencing earlier documents (usually functional user specifications like user scenarios, workflows, use cases, etc.)**
  - **Forward traceability – ensuring that each requirement can be uniquely identified (by name or identifier) for tracking through to design, testing, and implementation**

This is a key to success!

Traceability Matrix

Rqmt -> Source Doc	3.1.3	3.1.4	3.1.5	3.2.1	3.2.2	3.2.3
Workflow	WF 1			WF 2		
Use Case		UC 1			UC 3	
Scenario						SC 4
Strategic Plan			SC p.2			

Backward Traceability- requirement to source

Develop Testing Criteria

- Start with a requirement
- Develop reasonable pass/fail criteria
  - How will you know if the software is effective in meeting requirement?
  - How will you know if the software is ineffective in meeting the criteria?
  - Many requirements are not absolute – define your tolerances
  - Restate requirement if necessary
  - Develop mitigation if necessary

Developing Criteria

- **SPAM Example:**
  - **Spam is any email that the user does not need or want.**
  - **Requirement: The email system shall provide a spam filter. Implied criteria:**
    - **100% of bad email (spam) is blocked**
    - **100% of good email (not spam) is not blocked**
  - **The only way to ensure that no spam gets through is to block all email**
  - **The only way to ensure that all good email gets through is to not block any email**
- **It's not possible to do both!**

What's reasonable?

- **Unblocked Spam:**
  - **If 1 in 100 spam emails is not blocked, is that reasonable?**
  - **If 10 in 100 spam emails are not blocked, is that reasonable?**
- **Lost good email:**
  - **If 1 email is lost is that reasonable?**
  - **If 5 emails are lost is that reasonable?**
- **Quantify / assess the problem**
  - **Avg user receives 350 emails per week**
  - **Approximately 25% of incoming emails are spam**
  - **Problems due to a lost or missed emails are considered preventable and unacceptable**

Consider Risk

- Define the business issues and risks?
  - The time it takes for users to deal with spam impacts productivity and increases probability of errors
  - Loss of good email may or may not have serious consequences depending upon content.
- What steps can be taken to mitigate the threats/risks?
  - Provide a quarantine area for all blocked emails
  - Train users to identify and block spam
  - Train users to review quarantined email for good email

Finalize Requirements

- **Original Requirement: The email system shall provide a spam filter.**
- **New Requirements for the email system:**
  - **All email will be evaluated for spam prior to delivery**
  - **Likely spam will be quarantined automatically**
  - **Users can manually quarantine emails**
  - **Users can review quarantined emails**
  - **Users can un-quarantine good emails**
  - **Provide on-line help for managing spam**

Target Criteria

- **Target Criteria**
  - **100% if email will be evaluated for spam**
  - **Zero emails are irretrievably lost**
  - **No more than 1% of spam emails are not quarantined**
  - **User documentation, online help, and training materials include instructions for**
    - **quarantining spam email**
    - **reviewing quarantined email and un-quarantine good email to**

Mitigation

- All employees will be informed of email policies and procedures
- Employee orientation will include training to manage email
  - Identify and quarantine SPAM
  - Regularly review quarantined email
  - Identify good email and un-quarantine

Next...

- **Test Strategies and Planning**

Start your test plan early…