Phase – 1: Requirement Engineering

Requirement Engineering, Requirement Elicitation, Requirement Documentation Quality of requirements

Suleman Shahid

Slides based on

1

Poll

2

Requirement Engineering Process

LAHORE UNIVERSITY OF MANAGEMENT SCIENCES

3

Requirement Engineering

- Requirement elicitation and analysis
- Requirement Specification (Modeling)
- Requirement Validation
- Requirement Documentation

4

Requirement Engineering

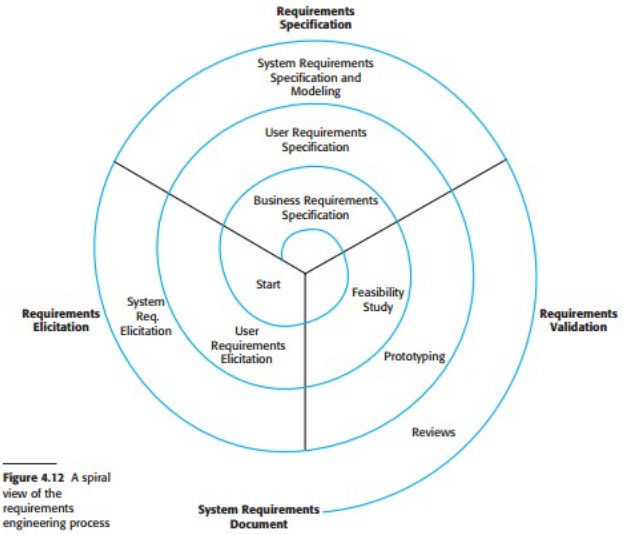


Figure 4.12 A spiral view of the requirements engineering process

5

1 – Requirement Elicitation

6

Requirement Elicitation – Sources

- Stakeholders
- Documents
  - Universal documents (standards, legal documents etc)
  - Domain documents
- Systems in operations
  - Legacy or predecessors and/or competing systems

Requirement Elicitation

- Don't Gather Requirements – Dig for Them
- Work with a User to Think Like a User
- Abstractions Live Longer than Details
- Use a Project Glossary

Elicitation Techniques

- Survey Techniques
- Creativity techniques
- Document-centric
- Observation based
- Support Techniques

Requirement Elicitation

- Survey Techniques
  - Interview
  - Questionnaires
- Creativity Techniques
  - Brainstorming/Brainstorming paradox
  - 6-3-5 (6p, 3i, 5f)
  - Early identification of risks
  - Six Thinking Hats – Change of perspective

Requirement Elicitation

Document-centric Techniques
- Reuse solutions and experiences
- System archaeology

- Observation Techniques
  - Contextual, Cognitive
  - Field observation (operational procedures)

Requirement Elicitation

- Support Techniques
  - Mind Mapping
  - Workshops/focus groups
  - CRC cards
  - Audio and video recording
  - Modeling action sequences
  - Prototypes for illustration

2 – Requirement Specification and Negotiation

Requirement Document

- Informal
  - Natural language
  - Structural natural language
- Semi-Formal
  - Modeling language
- Formal
- Hybrid

Requirement Document – Informal

- Natural Language

> 3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. (*Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.*)
>
> 3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. (*A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.*)

- Problems
  - ✧ Lack of clarity
  - ✧ Requirements confusion
  - ✧ Requirements amalgamation

Requirement Document – Informal

- Standard Natural language
  - Structured, form-based, tabular

Specification Document HMS

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

Structured Specification

Function Compute insulin dose: Safe sugar level.
Description Computes the dose of insulin to be delivered when the current measured sugar level is in
the safe zone between 3 and 7 units.
Inputs Current sugar reading (r2), the previous two readings (r0 and r1).
Source Current sugar reading from sensor. Other readings from memory.
Outputs CompDose—the dose in insulin to be delivered.
Destination Main control loop.
Action CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition r0 is replaced by r1 then r1 is replaced by r2.
Side effects None.

Structured Specification – Action

- CompDose is zero if the **sugar level is stable** or **falling** or if the **level is increasing but the rate of increase is decreasing.**
- If the **level is increasing** and the **rate of increase is increasing**, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result.
- If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered

Tabular Specifications

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 – r1) < (r1 – r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 – r1) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

Requirement Document – Semiformal

- Conceptual Models
  - Use Case
    - Overview of system functionality
  - Class Diagrams
    - Describes the static structure of the system
  - Activity Diagrams
    - Describes the dynamic behavior of the system
  - Statechart Diagram
    - Describes the dynamic behavior of the individual object

Requirement Document

- Formal
  - Mathematical specification
- Hybrid

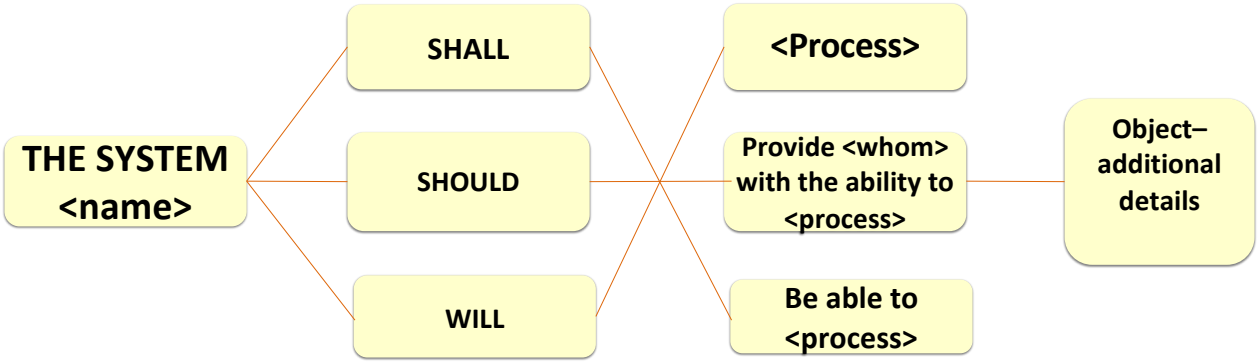
Documenting Requirements

Requirement Construction

- Activities of the system
  - **Autonomous system activity**: The system performs the process autonomously.
  - **User interaction**: The system provides the process as a service for the user.
  - **Interface requirement**: The system performs a process depending on a third party (e.g., another system). The system is passive and waits for an external event.
- Standards
  - IEEE format [IEEE Std. 830-1998]
  - User Stories

Requirement Construction – IEEE Standard

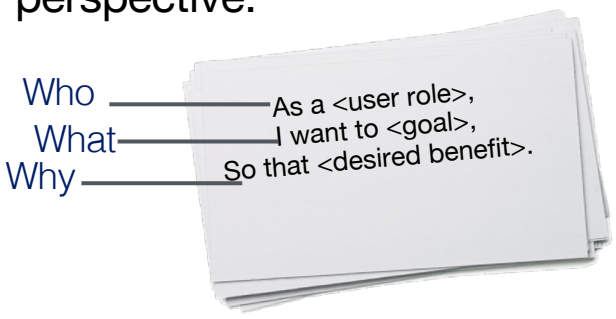
THE SYSTEM SHALL/SHOULD/WILL/ <process verb>
THE SYSTEM SHALL/SHOULD/WILL/ provide with the ability to <process verb>
THE SYSTEM SHALL/SHOULD/WILL/ be able to <process verb>



25

Requirement Construction – User Story

A brief, simple requirement statement from a user's perspective.



26

User Story

Title: Traveller wants to book a trip so that they can go to their destination

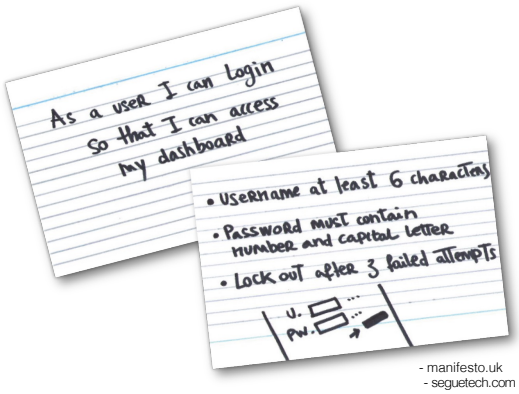
Story points: 3
Assigned to: Tom

Acceptance tests:
1. User can edit an airline booking
2. User can edit a car rental booking
3. User can edit a hotel booking
4. User can start editing from a screen that shows a booking

Other information:
- Attachments
- Screenshots
- UML
- Discussion
- Non-goals
- Additional details

Acceptance Criteria

- Acceptance Criteria as "Conditions that a software product must satisfy to be accepted by a user, customer or other stakeholder."
- Acceptance Criteria are a set of statements, each with a clear pass/fail result, that specify both functional and non-functional requirements



- manifesto.uk
- seguetech.com

Non-User Stories

- Technology foundation stories
  - At times these can be stated in customer terms
- Dependencies from external teams
- Creative elements
- Spikes
- Other types of stories... defects, maintenance, training, etc.

As a developer, I want to upgrade to the latest version of the database software so that we have a supported product

Spike: As a developer, I need to investigate a semantic search algorithm to facilitate natural language searching of the person's financial record.

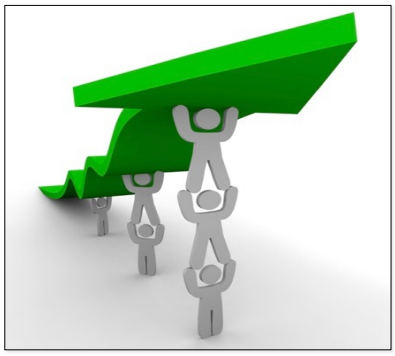
Who writes Requirements?

Traditional (IEEE Standard)

Driven from Product Owner

Agile (Story Writing) Everyone.

- Driven from Product Owner
- Assisted by the Team
- Requires Collaboration



Prioritizing the Product Backlog

- MoSCoW
  - Must – Should – Could – Won't
- Cost of Delay
- 100 Dollar Test
  - Cumulative voting
- KANO

KANO

The Kano Model is a product management prioritization framework that categorizes features based on their impact on customer satisfaction, separating them into Must-haves, Performers, Delighters, Indifferent, or Reverse attributes.

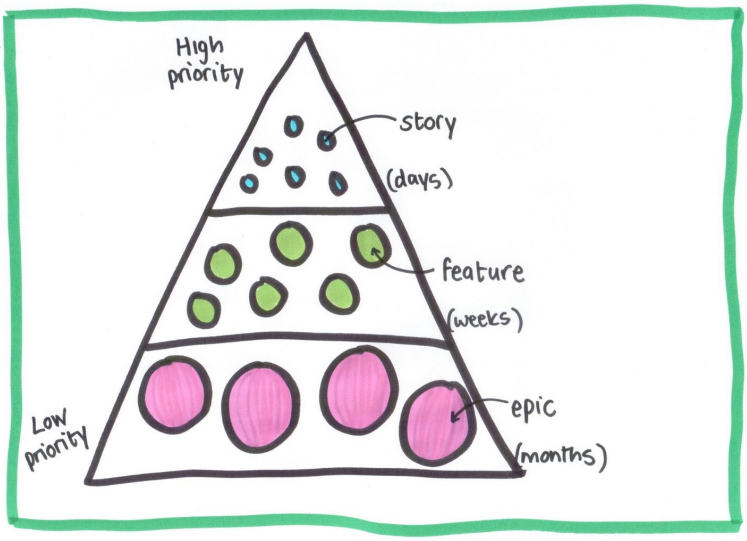
33

Requirement Categorization – Kano Model

- Dissatisfiers
  - Dissatisfiers are properties of the system that are self-evident and taken for granted
- Satisfiers
  - Satisfiers are explicitly demanded system properties
- Delighters
  - Delighters are system properties that the stakeholder does not know or expect and discovers only while using the system

34

Sub-Stories – Decomposing Stories



- leanpub.com

Slice the story

Once a story is close to being started, break it down.

"As an Authenticated Member, I want to log into the system so that my information can only be accessed by me."

...I want to log in with my username and password...

...I want to change my password...

...I want the system to warn me if my password is easy to guess...

...I want to be able to request a new password so that I am not locked out if I forget it

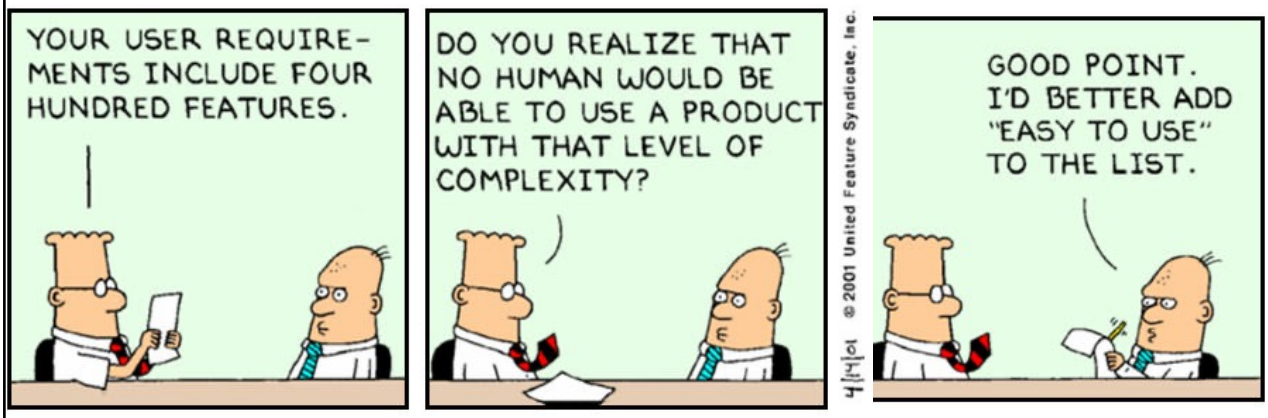
...I want to be notified if there have been three consecutive failed attempts to access my account...

- Mike Cohn

Documenting Requirements
Quality Criteria for Requirements

A Good Requirement Document

- Unambiguity and Consistency
- Clear
- Modifiable and extendable
- Complete
- Traceable



Qualities of Requirements

- Clear, understandable
  - Avoid unnecessary details. Should be concise, simple and precise
- Correct
  - Facts must be truthful
- Unambiguous
  - The system shall not accept passwords longer than 15
  - The system shall be implemented using ASP
- Verifiable
  - Avoid use of adverbs, adjectives

Examples

R – 003: The system should be user-friendly.

R – 003: The system should have an intuitive interface with clear navigation, consistent terminology, and interactive elements designed to facilitate a smooth user experience

R – 004: The software should be fast.

R – 004: The software should load a standard-sized dataset within 2 seconds and respond to user inputs with a maximum delay of 500 milliseconds

Qualities of Requirements

- Realizable
  - Feasible, possible
- Independent
- Atomic
- Traceable
- Necessary
  - Avoid unnecessary details
- Consistent
  - Non-conflicting

Examples

- Requirements should not contain unnecessary design and implementation information:

R – 005: Content *information shall be stored in a text file.*

Consistent

- There should not be any conflicts between the requirements. Conflicts may be direct or indirect. Direct conflicts occur when, in the same situation, different behavior is expected:

R – 006: Dates shall be displayed in the mm/dd/yyyy format.

R – 007: Dates shall be displayed in the dd/mm/yyyy format.

Conflicting Requirements

R – 007: Our application should be lightning-fast, with response times under one second for all user interactions. We need to impress users with seamless performance.

R – 008: Our application must be available 99.9% of the time. Users should access it anytime, anywhere. Downtime is unacceptable, even for maintenance.

Examples

- R – 009: Sometimes the user will enter Airport Code, which the system will understand, but sometimes the closest city may replace it, so the user does not need to know what the airport code is, and it will still be understood by the system.
- R – 009:The system shall identify the airport based on either an Airport Code or a City Name.

Examples

R – 010:The list of available flights shall include flight numbers, departure time, and arrival time for every leg of a flight.It should be sorted by price.

R – 010:The system shall display flight numbers, departure time, and arrival time for every leg of a flight in the list of available flights.

R – 011:The system shall provide an option to sort the list of available flights by price.

Examples

R – 012:The system shall resist concurrent usage by many users.

R – 012:The system shall support concurrent usage by up to 1000 users, maintaining response times below 5 seconds for standard transactions

R – 013:The system shall have a natural language interface that will understand voice commands given in any language.

R – 013:The system shall incorporate a natural language interface that supports voice commands in English and Spanish. The accuracy of voice command recognition should achieve a minimum of 95% accuracy in both languages, measured through comprehensive testing scenarios.

Examples

R – 014: The system shall provide the opportunity to book the flight, purchase a ticket, reserve a hotel room, reserve a car, and provide information about attractions.

R – 015: All requirements specified in the Vision document shall be implemented and tested.

R – 016: Payment by PayPal shall be available.
R – 017: Only credit card payments shall be accepted.

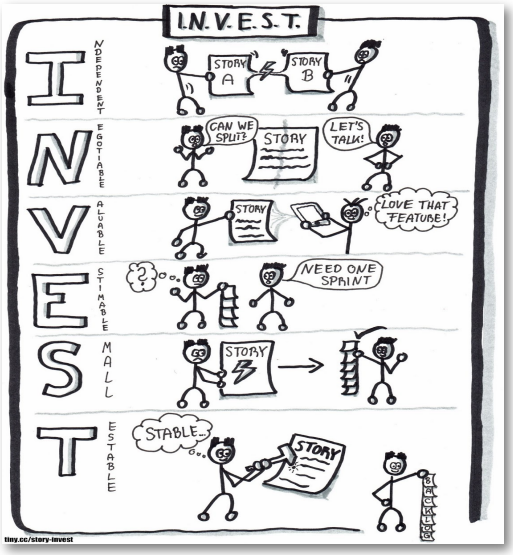
Examples

R – 0018: System should have a natural language interface.
R – 0019: System shall be developed in a month.

Qualities of a Good User Story

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

I.N.V.E.S.T.ing



- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

- Agile Planet

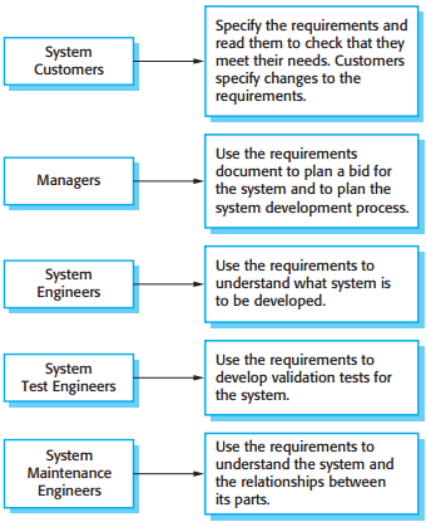
What to Watch Out For?

Mike Cohn's 'Catalog of Story Smells'

- Stories that are too small
- Stories too big....too many being split later
- Interdependent stories
- Too much detail
- Interface detail too soon
- Thinking too far ahead
- Lack of customer participation

Requirements Validation

Software Requirement Document



Thank you!