5. Requirement Engineering

Yan Shi

SE 273 Lecture Notes

Part of this note is adopted from Dr. Mike Rowe’s Lecture Notes
Requirements engineering process

- Feasibility Study
- Requirement Elicitation and Analysis
- Requirement Specification
- Requirement Validation
Feasibility studies

- A feasibility study decides whether or not the proposed system is worthwhile.

- A short (2-3 weeks) focused study that checks
  - If the system contributes to organisational objectives;
  - If the system can be engineered using current technology and within budget;
  - If the system can be integrated with other systems that are used.
What to evaluate?

- Economic feasibility
  - cost/benefit analysis
  - break even analysis
  - ROI

- Technical feasibility
  - Project size
  - Project structure
  - Familiarity with technology or application domain

- Operational feasibility
  - Will the project be deployed successfully in the organization?

- Schedule feasibility
  - estimation in Note 2!

- Legal and Political feasibility

http://www.umsl.edu/~sauterv/analysis/F08papers/Katimuneetorn_Feasibility_Study.html
Requirements engineering process

- Feasibility Study
- Requirement Elicitation and Analysis
- Requirement Specification
- Requirement Validation
Elicitation and analysis

- Involves technical staff working with customers to find out about
  - the application domain,
  - the services that the system should provide
  - and the system’s operational constraints.

- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called stakeholders.
Example: ATM stakeholders

- Bank customers
- Representatives of other banks
- Bank managers
- Counter staff
- Database administrators
- Security managers
- Marketing department
- Hardware and software maintenance engineers
- Banking regulators
Problems of requirements elicitation

Stakeholders don’t know what they really want.
Problems of requirements elicitation

Stakeholders have troubles expressing requirements
Problems of requirements elicitation

Other problems:

- Different stakeholders may have **conflicting** requirements.

- The requirements **change** during the analysis process. New stakeholders may emerge and the business environment change.
Elicitation and analysis process

- **Requirement discovery**
  - Interact with stakeholders to collect requirements
- **Requirement classification and organization**
  - Organize the collected requirements
- **Requirement prioritization and negotiation**
  - Prioritize requirements
  - Find and resolve conflicts
- **Requirement documentation**
  - Produce formal or informal documents

This may take several iterations:


(Each phase itself may also use several iterations)
Requirements discovery

- The process of gathering information about the proposed and existing systems and distilling the user and system requirements from this information.

- Sources of information include:
  - documentation
  - system stakeholders
  - specifications of similar systems

- Requirement discovery approaches:
  - interviewing
  - ethnography
  - scenarios
Interviewing

- In **formal or informal** interviewing, the RE team puts questions to stakeholders about
  - the system that they use and
  - the system to be developed.

- There are two types of interview
  - **Closed interviews** where a pre-defined set of questions are answered.
  - **Open interviews** where there is no pre-defined agenda and a range of issues are explored with stakeholders.
Interviews in practice

- Normally a mix of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- Interviews are not good for understanding domain requirements
  - Requirements engineers cannot understand specific domain terminology;
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn’t worth articulating.
Interviewers should be *open-minded*, willing to listen to stakeholders and should not have pre-conceived ideas about the requirements.

They should prompt the interviewee with a question or a proposal and should not simply expect them to respond to a question such as ‘what do you want’.
Exercise

- Each group prepare a list of 8 specific questions about a game in general.
- Each group is split into 2 smaller parts A and B.
- Group 1-A interview Group 2-B;...; Group N-A interview Group 1-B:
  - Start with the 8 questions
  - Continue to open discussions
  - Record the interview and pass it to the interviewee
Ethnography

- Observing and analysing how people actually work.
  - Observations
  - Contextual Interview

- Benefits:
  - People do not have to explain or articulate their work.
  - Social and organisational factors of importance may be observed.
  - Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.

- Issues:
  - Spy?? people may not be comfortable
  - cannot always identify new features to be added
  - not a complete elicitation method; used to serve other approaches (e.g., scenarios)
Scenarios

- Scenarios are *real-life examples* of how a system can be used.

- They should include
  - the starting situation;
  - the normal flow of events;
  - what can go wrong (optional);
  - A description of the state when the scenario finishes.
Types of Scenarios

- **As-is scenarios**
  - describes current situation

- **Visionary scenarios**
  - describes future system

- **Evaluation scenarios**
  - describes user tasks for evaluating the system (acceptance criteria)

- **Training scenarios**
  - introduces new users to the system
Heuristics for Identifying Scenarios

- **Interviews**: Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system? Who else can modify this data?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?

- **Task observation (Ethnography)** if the system already exists
  - Ask to speak to the end user, not just to the software contractor
  - Expect resistance and try to overcome it
Scenario Example: Warehouse on Fire

- **Bob**, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, **Alice**, reports the emergency from her laptop.

- Alice enters the address of the building, a brief description of its location (i.e., north west corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appear to be relatively busy. She confirms her input and waits for an acknowledgment.

- **John**, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.

- Alice received the acknowledgment and the ETA.
Observations about Warehouse on Fire

Scenario

Concrete scenario

- Describes a single instance of reporting a fire incident.
- Does not describe all possible situations in which a fire can be reported.
- Use multiple scenarios to describe multiple possibilities.

Participating actors

- Bob, Alice and John
- or use their roles
### Scenario Example: Warehouse on Fire

<table>
<thead>
<tr>
<th>Scenario name</th>
<th>warehouseOnFire</th>
</tr>
</thead>
</table>
| Participating actor instances | bob, alice:FieldOfficer  
| | john:Dispatcher |
| Flow of events | 1. Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the “Report Emergency” function from her FRIEND laptop.  
| | 2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene, given that the area appears to be relatively busy. She confirms her input and waits for an acknowledgment.  
| | 3. John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.  
| | 4. Alice receives the acknowledgment and the ETA.  |
Writing Scenarios

- noun verb direct-object
  - simple sentence
  - avoid passive voice

- No hypotheticals
  - no “if”, “when”, “or”, “until”
  - If the condition is critical, write a separate scenario for the alternative case.

- Document observables
  - Don’t write “Alice finds it acceptable”

- Give specific messages printed on the screen

- **Key: from the user’s perspective**
Exercise

- Write scenarios for an email system
  - Write a new email
  - Reply an email
  - Make an appointment
  - Search for an email
  - Create a new contact
Use case model

- Use-cases are a **scenario based technique** in the **UML**
  - identify the actors in an interaction
  - describe the interaction itself

- A set of use cases should describe all possible interactions with the system.

- use case = a named collection of scenarios
Use case name: ReportEmergency

Participating actors:

Flow of events:

Entry condition:

Exit condition:

Quality requirements:
Requirements Elicitation Using Use Cases

- Identifying Scenarios
- Identifying Use Cases
- Identifying Actors
- Refining Use Cases
- Identifying Relationships between Actors and Use Cases
- Identifying Nonfunctional Requirements
Identifying Use Cases

- Use Case
  - Specifies all possible scenarios for a given functionality
  - Initiated by an actor

- Motivations for use cases
  - Generalizing related scenarios help developers define the scope of the system
  - The role of each user of the system is clarified

- Use Case Descriptions
  - Entry and exit conditions
  - Flow of events
  - Quality requirements
Identifying Actors

- **Actors**
  - person or machine using the system in a particular role.

- Actors usually correspond to existing roles within the client organization

- **Guide Questions**
  - Which user groups execute the system’s main functions?
  - Which user groups perform secondary functions, such as maintenance and administration?
  - Which user groups are supported by the system to perform their work?
  - With what external hardware or software system will the system interact?
Actors vs. Stakeholders

- **Actors**
  - interact with the system
  - They might be humans or other systems

- **Stakeholders**
  - have some interest in the system
  - They include the users and many others, e.g., CEO who invests in a system to improve the business processes but never uses the system.
Relationships Between Actors and Use Cases

- *initiate*

- *participate*

- Determines access rights
  - Who can initiate a functionality
  - Who else is involved in this functionality
Formulating Use Cases

- **Step 1: Name the use case**
  - Use case name: *ReportEmergency*

- **Step 2: Find the actors**
  - Generalize the concrete names (“Bob”) to participating actors (“Field officer”)
  - Participating Actors:
    - *Field Officer* (Bob and Alice in the Scenario)
    - *Dispatcher* (John in the Scenario)

- **Step 3: Concentrate on the flow of events**
  - Use informal natural language
  - Number them to form a flow

- **Step 4: Describe entry and exit conditions**
- **Step 5: Describe exceptions**
- **Step 6: List non-functional requirements related to the use case**
Use Case Example: ReportEmergency

- **Use case name**: ReportEmergency

- **Participating Actors**:
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)

- **Exceptions**:
  - FRIEND notifies the FieldOfficer immediately if the connection between her terminal and the central is lost.
  - FRIEND notifies the Dispatcher immediately if the connection between any logged in FieldOfficer and the central is lost.

- **Flow of Events**: on next slide.

- **Special Requirements (Non-functional)**:
  - The system acknowledges the FieldOfficer’s report within 30 seconds.
  - The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.
1. The **FieldOfficer** activates the “Report Emergency” function of her terminal.

2. FRIEND responds by presenting a form to the officer.

3. The FieldOfficer fills the form, by selecting the emergency level, type, location, and a brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. The FieldOfficer submits the form after finishing it.

4. FRIEND receives the form and notifies the **Dispatcher**.

5. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the **OpenIncident** use case.

6. The Dispatcher selects a response and acknowledges the emergency report.

7. FRIEND displays the acknowledgment and the selected response to the FieldOfficer.
How to describe Exceptions?

- List all possible exceptions in the section of EXCEPTIONS

- For each exception:
  - Describe a flow of event separately
  - Include precondition, flow of event, post condition
Choose proper name: **verb noun**
- use verb phrases for the use case
- indicate user’s objective
- Name actors with noun phrases

- Clearly distinguish actors’ actions from system’s actions
- Use active voice to phrase steps in flow of events
- The relationship between steps should be clear
- Describe complete user transaction
- Describe exceptions separately
- Do not describe the user interface
- Use cases should not exceed 2-3 pages
  - break up using <<include>> and <<extends>> relationships
Exercise: Place Order

- Write a use case for a shopper to place order in an online shopping system.
  - note that the online shopping system is working together with an external billing system and fulfillment system.
Refining Use Cases

- **Goal:** completeness and correctness

- **Refinements**
  - Details of objects in the system
  - Low-level interactions between actors and system
  - Access rights
  - Missing exceptions
  - Common functionality among use cases
Defining Non-Functional Requirements

- Non-functional requirement: constraints and qualities.

- Run-time qualities:
  - Usability
  - Configurability and supportability
  - Correctness, reliability, availability
  - Performance
  - Security and fault tolerance
  - Scalability

- Development-time qualities:
  - Localizability
  - Extensibility
  - Reusability

- Constraints
  - Implementation
  - Interface
  - Operations
  - Packaging
  - Standard
  - Legal
Nonfunctional Requirements: Trigger Questions

- **User interface and human factors**
  - What type of user will be using the system?
  - Will more than one type of user be using the system?
  - What sort of training will be required for each type of user?
  - Is it particularly important that the system be easy to learn?
  - Is it particularly important that users be protected from making errors?
  - What sort of input/output devices for the human interface are available, and what are their characteristics?

- **Documentation**
  - What kind of documentation is required?
  - What audience is to be addressed by each document?

- **Hardware considerations**
  - What hardware is the proposed system to be used on?
  - What are the characteristics of the target hardware, including memory size and auxiliary storage space?
Nonfunctional Requirements: Trigger Questions 2

- **Performance characteristics**
  - Are there any speed, throughput, or response time constraints on the system?
  - Are there size or capacity constraints on the data to be processed by the system?

- **Error handling and extreme conditions**
  - How should the system respond to input errors?
  - How should the system respond to extreme conditions?

- **System interfacing**
  - Is input coming from systems outside the proposed system?
  - Is output going to systems outside the proposed system?
  - Are there restrictions on the format or medium that must be used for input or output?
Nonfunctional Requirements: Trigger Questions 3

- **Quality issues**
  - What are the requirements for reliability?
  - Must the system trap faults?
  - What is the maximum time for restarting the system after a failure?
  - What is the acceptable system downtime per 24-hour period?
  - Is it important that the system be portable (able to move to different hardware or operating system environments)?

- **System Modifications**
  - What parts of the system are likely candidates for later modification?
  - What sorts of modifications are expected?

- **Physical Environment**
  - Where will the target equipment operate?
  - Will the target equipment be in one or several locations?
  - Will the environmental conditions in any way be out of the ordinary (for example, unusual temperatures, vibrations, magnetic fields, ...)?
Security Issues

- Must access to any data or the system itself be controlled?
- Is physical security an issue?

Resources and Management Issues

- How often will the system be backed up?
- Who will be responsible for the back up?
- Who is responsible for system installation?
- Who will be responsible for system maintenance?
How to Specify a Use Case
(Summary)

- Name of Use Case
- Actors
  - Description of Actors involved in use case
- Entry condition
  - “This use case starts when…”
- Flow of Events
  - Free form, informal natural language
- Exit condition
  - “This use case terminates when…”
- Exceptions
  - Describe what happens if things go wrong
  - Has its own flow of events!
- Non-Functional Requirements
  - Qualities and Constraints
Use case diagram

- **Use case:**
  - A written description of one use of the system.
    - Who will use the system?
    - What will they be able to do with it?
    - No standard for format and content.

- **Use case diagram**
  - A graphical representation of a use case.
  - A diagram type defined in UML.
  - Gives the developers and users a high-level view of the relationships among the different Use Cases and Actors of a system.

- **UML: Unified Modeling Language**
Use Case Diagram

- Four types of components:
  - Actors: stick figure
  - Use case: oval
  - The system: a boundary box
  - Relationships: lines
Draw a Use Case Diagram

- For a use case: all participants (actors) are associated with it.

- Use non-directed lines to connect Actors and Use Cases.
Draw a Use Case Diagram

- For a system: list all use cases and their actors

- Put all use cases inside the system boundary box, all actors outside the box. (Optional)
- Put primary actors (initiators) on the left, secondary actors (participants) on the right.
- Each use case should have a text description (as described previously).
- **<<include>>:**
  - If multiple use cases share the same partial flow of events, make the partial flow a separate use case and include it in these multiple use cases.
  - OR the inclusion use case is an important part of the base use case.
    - The base case is incomplete without the inclusion case.

- **<<extend>>:**
  - The extension use case consists of additional behavior that can incrementally augment the behavior of the base use case.
    - The extension use case is not meaningful on its own.
Example

http://creately.com/blog/diagrams/use-case-diagram-relationships/
YOU BOTH HAVE SUBMITTED A BUNCH OF NEW REQUIREMENTS, BUT WE CANNOT IMPLEMENT THEM ALL. WHAT ARE THE LEAST IMPORTANT ONES?

OK, NO PROBLEM. WE'LL IMPLEMENT ALL OF THEM AND JUST FOREGO TESTING.

THE CONSULTANTS HANDBOOK PART 9: THE CUSTOMER IS KING
See on the course webpage:
- the requirement specification document template for this project (Phase 4)
Requirements engineering process

- Feasibility Study
- Requirement Elicitation and Analysis
- Requirement Specification
- Requirement Validation
Without Requirement Validation...

Our sales guy vastly underbid a job. Now it's my project to install the system in a way that's profitable.

Blame your customer for underspecifying the features then charge her through the nose for change orders.

Three million dollars for an electrical plug?

The base model uses a potato battery.

© Scott Adams, Inc./Dist. by UFS, Inc.
Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.

- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

- SMART requirement!
- Focus: correctness and completeness
Requirements validation techniques

- **ad-hoc technique**
  - a non-systematic way of identifying defects. Just read through an artifact.

- **checklist-based technique**
  - the inspectors are provided with a list of general defect classes to check against. Check each item in an artifact to make sure that it is consistent with the checklist.
  - Example: Programming Ground Rules.

- **Review**
  - a manual process that involves multiple readers checking document for anomalies and omissions.

- **Inspection and Walkthrough**
  - a peer group review: review a software document line by line or follows the logical flow of the document (code) while in the same room.
- **scenario-based technique**
  - the use of scenarios to guide inspectors on how to find required information as well as what that information should look like

- **perspective-based reading**
  - one step deeper than scenario-based, various people read the (requirements) document from a particular point of view according to the perspective represented by different stakeholders in the project.
  - Role playing
Requirements validation techniques

- **Prototyping**
  - Using an executable model of the system to check requirements.

- **Test-case generation**
  - Developing tests for requirements to check testability.
Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated.

- The review team:
  - review leader
  - producer
  - recorder
  - reviewers

- Reviews may be formal (with completed documents) or informal.

- Good communications between developers, customers and users can resolve problems at an early stage.
Guidelines for Effective Review

- Review the product, not the producer
- Identify problems, don’t try to solve them
- Limit debates
- Take written notes
- Use a checklist
- Allocate and schedule time for reviews
- Have an agenda and stick to it
- Keep the review session to 1-2 hours
- Keep the review team to 3-6 participants
Review: SMART Requirements

Mannion and Keepence, 1995

- **Specific:**
  - clear without ambiguity;
  - consistent terminology;
  - simple, no double requirements X and Y;
  - of appropriate level of details

- **Measurable:**
  - possible to verify the requirement is met

- **Attainable:**
  - technically feasible

- **Realizable:**
  - realistic given the resources

- **Traceable:**
  - linked from its conception through its specification to its subsequence design, implementation and test.

- And more... (correctness, completeness)
Prototyping: Why?

- Ch. 17 introduction and 17.4
- Two goals for software development:
  1. Getting the right software. (requirements)
  2. Getting the software right. (design, implementation)
- Problem:
  - (2) takes too much time,
  - user often can't tell if got (1) done until (2) is done
- Solution: rapid prototyping
Prototyping: How?

- Goal: quickly generate something that can be tested by user
  - no design, no documentation
  - minimal implementation
  - fake calculations
  - assume input in simplistic format, etc.
  - no test plan
  - test with users: talk them through the form
  - throw away after it is used: not delivered or reused!
Prototyping: Issues?

- Prototype can be expensive to build
  - If not careful, will implement major features of system → this could be wasted work!
  - User may have very different idea than you of what the fields will do → ends up being an ineffective test
  - In general, often perceived as more work than necessary

- Solution: paper prototypes ("puppet shows")
  - Lumzy: http://lumzy.com/app/
Prototyping: More?

- Prototyping is not just for requirement validation...
  - requirement elicitation
  - try out design ideas
How to ensure traceability

- Organize your requirements well and use labels that contain more information than just a number.
- Use the labels in other artifacts to link them to the requirement.
- Possible use traceability matrices that show where a requirement is being referenced.

<table>
<thead>
<tr>
<th>Rqmt</th>
<th>Test</th>
<th>Code Class</th>
<th>Etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-CAP-42.1</td>
<td>Test-CAP-1234</td>
<td>Users</td>
<td>...</td>
</tr>
<tr>
<td>·</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>·</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>·</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Never change a requirement label once it is created.
  - Don’t use auto-numbering!
  - If you do so, make sure you also use cross-reference!
Summary

- Requirements engineering process
  - Feasibility study
  - Requirement elicitation and analysis
    - Discovery, Organization, Prioritization and Documentation
    - Interviews
    - Scenarios
  - Requirement specification
    - Use case model
    - Use case diagram
  - Requirement validation
    - Reviews
    - Prototyping