Domain Understanding and Requirements Elicitation

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada
The Requirement Engineering process

1. Domain understanding and elicitation
2. Evaluation and agreement
3. Specification and documentation
4. Validation and verification

The entire process can (and usually is) repeated many times!
Types of Projects

- **Rabbit** - the most agile project
  - frequent iterations, each iteration delivers a small increment to the working functionality
  - *learning* but not much writing (official) requirements
  - using *scenarios* and *prototypes* for delivering the requirements to developers
  - have relatively short live

- **Horse** - fast, strong, dependable
  - most common corporate projects
  - there is a need for some documentation as requirements are shared between departments
  - a few stakeholders $\implies$ consistent documentation
  - some degree of formality in the development process
  - medium longevity
Types of Projects

- **Elephant** - solid, strong, long life and long memory
  - a need for a *complete* requirements specification
  - contractual arrangements as outsourcing
  - often the regulators demand not only full specification but also a detailed description of the process (so they can audit it)
  - many stakeholders and developers
  - **requirements extremely important**
  - formal methods are often used
Objectives and Knowledge Acquisition

- **Objectives**
  - Understand the system-as-is and its context
  - Identify the problems and opportunities calling for a new system
  - Discover the *real* needs of stakeholders with respect to the new system
  - Explore alternative ways in which the new system could address those needs.

- **Knowledge Acquisition**
  - Knowledge about the *organization* - its structure, business objectives, policies, roles and responsibilities
  - Knowledge about the *domain* in which the problem world is rooted - the concepts involved in this domain, the objectives specific to it, the regulations that may be imposed on it.
  - Knowledge about the *system-as-is* - its objectives, the actors (i.e. active entities) and resources involved the tasks and workflows, and the problems raised in this context.
Identifying stakeholders and interacting with them

- **Stakeholder analysis**
  - Relevant position in the organization
  - Effective role in making decision about the system-to-be
  - Level of domain expertise
  - Exposure to perceived problems
  - Influence in system acceptance
  - Personal objectives and conflict of interests
Identifying stakeholders and interacting with them

- Handling obstacles to effective knowledge acquisition
  - Distributed and conflicting knowledge sources
  - Difficult access to sources
  - **Obstacle to good communication** - important, often underestimated!
  - *Tacit knowledge and hidden needs* - important, difficult to get!
  - Socio-political factors
  - Unstable conditions

- Interacting with stakeholders - **difficult, important, often underestimated**!
  - Communication skills
  - Knowledge reformulation
  - *Culture barriers* (i.e. nurses vs information technology people, etc.)
Elevator System

Example (More Specific Case)

- Assume the system-to-be is an elevator system in 15 floors five stars hotel with an outdoor swimming pool on the last floor, two restaurants on 14 floor, spa, gym and indoor pool on floor -1, shops and reception on ground floor, conference rooms on floor 2 and parkings on floors -2 and -3.
- The chain does not have a similar hotel, the closes one that can be used as a system-as-is has only 6 floors, indoor swimming pool on 6th floor, restaurant on ground floor, gym and spa in the basement and parking is outdoor.
Example (Stakeholders Identification)

Stakeholders are people, groups, organizations that might be affect by the outcome of a project. At the hotel, the installment of new elevator system can result in some massive changes in terms of management. Hence the stakeholders include:

- **Hotel Owner/manager** the person who will be approving the new project
- **The architect of the building** An architect plays a vital role in installing the elevator system in any hotel or a building. They are helpful providing with the current status of the load on an elevator and can tell the maximum amount weight a single elevator could carry without harming the structure or the tower.
- **Restaurant manager** find out the estimated traffic for the restaurant floor
Example (Stakeholders Identification)

Stakeholders are people, groups, organizations that might be affected by the outcome of a project. At the hotel, the installation of a new elevator system can result in some massive changes in terms of management. Hence the stakeholders include:

- **Front desk / Community Relations Manager / Business analysts** knows how many people would check in every day and what the peak season occupancy is
- **Head maid** knows the times the maids need the lift
- **Restaurant manager** knows the times the restaurant workers need the elevators
- **Security head / Elevator repair contractor / Maintenance manager** knows the relevant safety details for the elevator system
Artifact-driven elicitation techniques

- Background study
- Data collection, questionnaires
- **Scenarios**, storyboards for problem world exploration
- Prototypes, mock-ups for early feedback
Background study

- Collect, read, synthesize documents about:
  - the organization: organizational charts, business plans, financial reports, meeting minutes, etc.
  - the domain: books, surveys, articles, regulations, reports on similar systems in the same domain.
  - the system-as-is: documented workflows, procedures, business rules; exchanged documents; defect/complaint reports, change requests, etc.

- Provides basics for getting prepared before meeting stakeholders \(\implies\) prerequisite to other techniques.
- Data mining problem: huge documentation, irrelevant details, outdated info.
- Solution: use meta-knowledge to prune the document space.
  - know what you need to know and what you don’t need to know.
Data collection

- Gather undocumented facts and figures
  - marketing data, usage statistics, performance figures, costs, etc.
  - by designed experiments or selection of representative data sets from available sources (use of statistical sampling techniques)
- May complement background study
- Helpful for eliciting non-functional requirements on performance, usability, cost etc.
- Difficulties:
  - Getting reliable data may take time
  - Data must be correctly interpreted
Questionnaires

- Submit a list of questions to selected stakeholders, each with a list of possible answers (+ brief context if needed)
  - Multiple choice question: one answer to be selected from answer list
  - Weighting question: list of statements to be weighted... qualitatively ('high', 'low', ...), or quantitatively (percentages) to express perceived importance, preference, risk etc.

- Effective for acquiring subjective info quickly, cheaply, remotely from many people
- Helpful for preparing better focused interviews
Scenarios and storyboards

- **Probably the most important technique!**
- **Goal:** acquire or validate info from concrete examples through narratives
  - how things are running in the system-as-is
  - how things should be running in the system-to-be
- **Storyboard:** tells a story by a sequence of snapshots
  - Snapshot = sentence, sketch, slide, picture, etc.
  - Possibly structured with annotations:
  - WHO are the players, WHAT happens to them, WHY this happens, WHAT IF this does / does not happen, etc
  - *Passive* mode (for validation): stakeholders are told the story
  - *Active* mode (for joint exploration): stakeholders contribute
Scenarios

- Illustrate typical sequences of interaction among system components to meet an implicit objective
- Widely used for:
  - explanation of system-as-is
  - exploration of system-to-be + elicitation of further info ... e.g. WHY this interaction sequence? WHY among these components?
  - specification of acceptance test cases
- Represented by text or diagram (formal or informal)
Scenario example: meeting scheduling

1. The initiator \textit{asks} the scheduler for planning a meeting within some date range. The request includes a list of desired participants.
2. The scheduler checks that the initiator is entitled to do so and that the request is valid. It \textit{confirms} to the initiator that the requested meeting is initiated.
3. The scheduler \textit{asks} all participants in the submitted list to send their date and location constraints back within the prescribed date range.
4. When a participant \textit{returns} her constraints, the scheduler validates them (e.g., with respect to the prescribed date range). It \textit{confirms} to the participant that the constraints have been safely received.
5. Once all valid constraints are \textit{received}, the scheduler determines a meeting date and location that fit them.
6. The scheduler \textit{notifies} the scheduled meeting date and location to the initiator and to all invited participants.
Types of scenario

- **Positive** scenario = one behavior the system should cover (an example)
- **Negative** scenario = one behavior the system should exclude (a counter-example), e.g.
  1. A participant returns a list of constraints covering all dates within the given date range
  2. The scheduler forwards this message to all participants asking them for alternative constraints within extended date range
- **Normal** scenario: everything proceeds as expected
- **Abnormal** scenario = a desired interaction sequence in exception situation (still positive)
  e.g. meeting initiator not authorized participant constraints not valid
Example (Positive or Normal Scenario)

1. The user requests for an up-elevator from the ground floor by pressing the button. Up button illuminates.

2. The elevator visits the ground floor and the doors open. The up button’s illumination is cancelled. The user steps into the elevator. The doors close.

3. The user requests to go to the 10th floor by pressing button 10. The button 10 illuminates.

4. The elevator visits the 10th floor and the door open. The button 10’s illumination is cancelled the user steps out of the elevator. The door close.
Example (Negative Scenario)

1. The user requests for an up-elevator from the ground floor by pressing the up button. Up button illuminates.
2. The elevator visits the ground floor and the door open. The up button’s illumination is cancelled. The user steps into the elevator. The door close.
3. The user request to go to the 10th floor by pressing button 10. The button 10 illuminates.
4. The elevator visits the 9th floor and the doors open. The button 10th illumination is cancelled. The user step out of the elevator. The doors close.
Example (Abnormal Scenario)

1. The elevators must switch to “out of service” in the case of fire. Only firemen should be able to operate the elevators in this condition.

2. The user requests for an elevator from the 10th floor by pressing the up button. Up button illuminates.

3. The elevator visits the ground floor and doors open. The Up buttons illumination is cancelled. The user step into the elevator. There is already a second user in the elevator with the button 13 illuminated. The door closed.

4. The first user requests to go to ground floor by pressing button G. The button G illuminates.

5. The elevator visits the 13th floor and the doors open. The button 13th illumination is cancelled. The second user step out of the elevator when the doors close.

6. The elevator visits the ground floor and the doors open. The button G’s illumination is cancelled. The first user steps out of the elevator when the doors close.
Scenarios: pros & cons

😊 Concrete examples/counter-examples
😊 Narrative style (appealing to stakeholders)
😊 Yield animation sequences, acceptance test cases
😊 Inherently partial (cf. test coverage problem)
😊 Combinatorial explosion (cf. program traces)
😊 Potential overspecification: unnecessary sequencing, premature software-environment boundary
😊 May contain irrelevant details, incompatible granularities from different stakeholders
😊 Keep requirements implicit

    cf. confidentiality req in negative scenario example

Concrete scenarios naturally jump in anyway...

    invaluable as initial elicitation vehicles
Prototypes

- **Goal**: check req adequacy from direct user feedback, by showing reduced sketch of software-to-be in action
  - focus on unclear, hard-to-formulate requirements to elicit further
- **Prototype** = quick implementation of some aspects
  - **Functional** prototype: focus on specific functional requirements
    e.g. initiating meeting, gathering participant constraints
  - **User interface** prototype: focus on usability by showing input-output forms, dialog patterns
    e.g. static/dynamic interaction to get participant constraints
- Quick implementation: by use of very high-level programming language, executable spec language, generic services, etc.
Prototypes: pros and cons

😊 Concrete flavor of what the software will look like
=> clarify reqs, elicit hidden ones, improve adequacy, understand implications, ...

😊 Other uses: user training, stubb for integration testing, ...

😊 Does not cover all aspects
- missing functionalities
- ignores important non-functional reqs (performance, cost, ...)

😊 Can be misleading, set expectations too high

😊 'Quick-and-dirty' code, hard to reuse for sw development

😊 Potential inconsistencies between modified code and documented reqs