Contents

Requirements Engineering

- These cover not only the functionality of a system, but also:
  - Non-functional issues (e.g. performance, relatibility, etc.)
  - Constraints on design (e.g. must operate with current software/hardware)
  - Constraints on implementation (e.g. must be written in Java)

The Elements of Requirements Engineering

- Since the process and outcome must be repeatable, we need a fixed process
  - Elicitation - How do we extract the requirements?
  - Analysis
  - Representation - English, Diagrams, Mathematics
  - Validation
- This is not a lineas process.
- Humans can be ambiguous, and the process is imperfect
- We loop through the process until we're done, and at the end of it we have a requirements document, which is handed off to the developers, which they use to verify that they're meeting the requirements
- When the project is complete, the client then uses to verify that it's been done up to the requirements

Stakeholders

- There are two main stakeholders in requirements elicitation
  1. The customer - these are the targets for requirements elicitation and will generally only communicate with the domain-specific knowledge
  2. The developer - these need to know the constraints

Elicitation Techniques

- All of these techniques have positives and negatives
- Verbal techniques can be ambiguous, while written techniques can be problematic for non-native speakers
- Quentionnaires
- Interviews
- Focus groups and workshops

- Naturalistic observation
- Studying documentation

Quality Requirements

- Requirements should be
  - Non-amibguous, so that behaviour of the system can be clearly determined
  - Quantifiable and testable, e.g. "it should have good usability" isn't testable
  - Traceable, they can be traced back for justification or qualification
  - Atomic, not combining a number of requirements into a single requirement - look out for *and* and *but*
  - Independent, each requirement should be verifiable as done or not without comparing or linking it to other requirements
    - Complex requirements should be broken down into constituent requirements
  - Feasible, should be possible to complete in a reasonable amount of time and actually necessary
  - Prioritised, do the important things first even if they're a minority of the problem

Representing Requirements

- *A software Requirements Specification (SRS)* is a description of a system to be developed.
- It contains
  - Functional requirements
  - Non-function requirements
  - Constraints
  - External Interface Requirements (humans, machines, …)

Techniqus for Writing Down Requirements

- Work Breakdown Structure
  - Example: Construction of a House
    - Internal
      - Electrical
      - Plumbing

        * Foundation
            · Excavate
            · Steel Erection
        * External
            · Masonry Work
            · Building Finishes

- Use Cases
    - Invented by Ivar Jacobson in the 1960's
    - Adopted by OOP community in 1990
- Data Flow Diagrams, Activity Diagram, etc..

What is a Use Case

> A use case is a description of the possible sequences of interactions between the system under discussion and its external actors, related to a particular goal

- A thought capturing format of writing
- Documents the *behaviour* of the system
- Associated with the goal of **one particular actor* called *primary actor*
- Collections together
    - All scenarios related to *that* goal of *that* primary actor
    - Including those where goal is achieved
    - Including those where goal is not achieved
- An actor may be
    - A person
    - A group of people
    - A computer system
    - A component
    - ...

Stages in Use Case Writing

- Actors and Goals
- Definition of success
- Failure conditions
- Failure handling

Tolerance and Hardness

- No two people have *exactly* the same style
- High ceremony vs low ceremony
  - Medial devices, power plants, vehicles, …
  - Small websites, point-of-sale software, …
- Use appriopriate template

Basic Concepts

Actors

- Anything that has behaviour
- System Under Discussion (SuD)
- Subsystem of SuD
- Primary Actor
- Secondary Actor(s)

Goals

- Functional scope
  - Create a goal-list
  - Decide whether it is “in-scope” or “out-of-scope”
- Design Scope
  - Boundary of the organisation
  - Boundary of the system
  - Interfaces to secondary actors

Actors Have Goals

- A goal not related to any primary actor will not get done
- An actor achieves a goal through interations
  - Simple interact (e.g. message, method-call, etc.)
  - Sequence of interations
  - A set of sequences

Scenarios

- Conditions
  - Pre-conditions for the goal (e.g. trigger, other goals)
  - Post-conditions for the goal
- Goal success (Basic Flow)
- Goal Failure (Alternative Flow)

A scenario is a sequence of interactions that happens under certain conditions, with the intent of achieving the primary actor's goal. The interactions start from the triggering action and continue till the goal is achieved or abanonded

Use Case in Details

- A scenario consists of steps
- Example
  - An interaction between two actors
  - A validation
  - An internal state change
- The SuD must staisfy and protect the interests of all agreed-upon actors
- Scenario ends when all interests of actors are satisfied or protected

Levels of Goals

- Very high-level (organisation wide)
- High-level (business, system, strategic)
- Low-level (user)
- Sub-function: Usually exeedingly simple (log-on)

Writing a low-level Use Case

- Each step from a high-level use case can be a goal of a low-level use case
- Steps for low-level use cases are usually actions
- One low-level use case can call/refer to another low-level use case, if necessary

Pre-condiyion

- Pre-condition: Assertions about the state of the world before use case can be triggered
  - Use is logged on, disk has sufficient space
- Bad pre-condition: conditions normally true, but cannot be guaranteed
  - Before transferring a file, user has saved latest copy

Success End Condition

- Goals of all actors (primary and/or secondary) have been achieved
- Assertion about state of the world at moment of successful competition
  - File saved in dropbox

Failure End Condition

- Many ways to fail
- Usually described with a conditional
  - If ATM did not despence cash, user's account is not debited

Extensions

- Brainstorm all conveivable failures
- List alternate success paths
- Evaluate, eliminate, merge ideas

Typical Reason for Failures

- Primary actor sends bad data or requests
- Validation checks not passed
- Secondary actors do no respond or fail
- Inaction by primary actor
- Bad data discovered inside SuD

Documenting Alternate Flows

- Use step number of main success flow where alternate can occur
- Examples
  - 2b: Network Down
  - 2c: Insufficient Funds
- Scenario ends in one of three ways:
  - Failure condition is fixed
  - Failure condition causes scenario to restart
  - Failure condition causes scenario to reboot