Wednesday June 15, 2016

Requirements Analysis

* Understanding what the stakeholders want
* One of the most difficult parts of software engineering...
  + Why?
* **Impact**: mistakes here can obviate the work products at any point in the process

Modeling Functional Requirements

* Useful when
  + System has a lot of services/function requirements
  + And/or a lot of actors involved in the functions
* Not useful when
  + System has few functional requirements (rarely the case)
  + And/or few actors/interfaces

Why Model Requirements

* Can reveal gaps of unknown or ambiguous functions
* Can show inconsistent or conflicting behaviors/constraints
* Restating the requirements in different form strengthens understanding and forces validation
* Lots of different model notations

Entity Relationship Diagram (ERD)

* Structural model of entities, actions, and relationships
* Boxes represent entities (external and data objects)
* Diamonds represent functional requirements (actions) OR relationships between objects
* Useful object attributes listed above the object
* For relationships, can show multiplicity (i.e., 1 to many)

Message Sequence Chart

* Scenario model showing a sequence of actions/events between entities for a particular scenario/flow (normal vs. exceptional)
* Helpful for analyzing the steps of functional requirements and which entities are involved in a particular situation
* Rectangles are entities, horizontal arrows are messages
* Dash arrow creates entity, zig zag shows time passing
* Hexagons are important object states (conditions)

State Machine

* Model showing different states of a particular entity or data object in the system and which behaviors cause a change in state
* State is a value for a set of attributes (i.e., isVisible, coverageActive, propertSold)
* Note: Makes a great basis for testing
  + Restricts all the specific paths that allow this data object to revolve (refer to example in powerpoint Lecture5.pdf)

Data Flow Diagram (DFD)

* Models the flow/transformation of data between functions
* Bubble is a process/function
* Arrow is data flow
* Rectangles are actors (external entities)
* Parallel lines are data stores (where data is fetched/saved)

Non-graphical Models

Several different formals for detailing:

* Functional requirements
* Constraints
* Entities (external and data objects) involved
* Relationships between them all

Use Case

* A plain language format for specifying requirements
* Something that an actor wants/needs to do with the software
* **Actor** – an entity external to the system that interacts with the system for the particular use/behavior to be described.

Analysis Patterns

Provide reusable solutions for requirements analysis for certain application domains

* Already have common requirements for project
* Cite advantages and limitations
* Suggest transformation, but not a perfect and brainless solution

Simple, Useful Patterns

* **CRUD**: create, read, update, delete
* **SCRUD**: search + CRUD
* Access control among user types implies
  + A User data object
  + A non-authenticated user type
  + Authentication functional requirement
  + User SCRUD

Requirements Engineering Summary

* All about understanding what the client wants and preparing for Planning and Design
* Talk to all Stakeholders and get their requirements
* Condense and visually model the requirements. Check for and resolve conflicts, errors and omissions
* Validate with Stakeholders (SHs must approve the specs)

Specifications

* The work product from Requirements Engineering
* A concise description of the system the engineers will build based on their understanding of what the client/users want
* Like processes, there are many different standards and variations for specifications; choose the best tool for the job
  + i.e., user stories are the specifications for at least a few Agile Methods

Validation and Verification

* Validate that the requirements are what the client wants
* Verify that the specifications correctly describe the requirements
* Verify the design correctly satisfies the specifications
* Verify the implementation correctly expresses the design
* **Result**: a correct software product

Validating Requirements

* Check with customer stakeholders to see if you are building the right system
* How to validate?
  + Read the requirements definition
  + Interviews
  + Walkthroughs
  + Reviews, formal inspections
  + Models, scenarios
  + Prototypes, simulations