Applied Software Project Management

Requirements

Software Requirements

- *Software requirements* are documentation that completely describes the behavior that is required of the software-before the software is designed built and tested.
  - Requirements analysts (or business analysts) build software requirements specifications through *requirements elicitation.*
    - Interviews with the users, stakeholders and anyone else whose perspective needs to be taken into account during the design, development and testing of the software
    - Observation of the users at work
    - Distribution of discussion summaries to verify the data gathered in interviews

Discussion Summary

- A requirements analyst can use a *discussion summary* to summarize information gathered during elicitation and validate it through a review.
- Notes gathered during the elicitation should fit into the discussion summary template
- The discussion summary outline can serve as a guide for a novice requirements analyst in leading interviews and meetings

Discussion Summary outline

1. Project background
   a) Purpose of project
   b) Scope of project
   c) Other background information
2. Perspectives
   a) Who will use the system?
   b) Who can provide input about the system?
3. Project Objectives
   a) Known business rules
   b) System information and/or diagrams
   c) Assumptions and dependencies
   d) Design and implementation constraints
4. Risks
5. Known future enhancements
6. References
7. Open, unresolved or TBD issues

Use Cases

- A *use case* is a description of a specific interaction that a user may have with the system.
- Use cases are deceptively simple tools for describing the functionality of the software.
  - Use cases do not describe any internal workings of the software, nor do they explain how that software will be implemented.
  - They simply show how the steps that the user follows to use the software to do his work.
  - All of the ways that the users interact with the software can be described in this manner.

Functional Requirements

- *Functional requirements* define the outward behavior required of the software project.
  - The goal of the requirement is to communicate the needed behavior in as clear and unambiguous a manner as possible.
  - The behavior in the requirement can contain lists, bullets, equations, pictures, references to external documents, and any other material that will help the reader understand what needs to be implemented.

Nonfunctional Requirements

- *Nonfunctional requirements* define characteristics of the software which do not change its behavior.
  - Users have implicit expectations about how well the software will work.
  - These characteristics include how easy the software is to use, how quickly it executes, how reliable it is, and how well it behaves when unexpected conditions arise.
  - The nonfunctional requirements define these aspects about the system.
    - The nonfunctional requirements are sometimes referred to as “non-behavioral requirements” or “software quality attributes”

Software Requirements Specification

- The *software requirements specification* (SRS) represents a complete description of the behavior of the software to be developed.
- The SRS includes:
  - A set of use cases that describe all of the interactions that the users will have with the software.
  - All of the functional requirements necessary to define the internal workings of the software: calculations, technical details, data manipulation and processing, and other specific functionality that shows how the use cases are to be satisfied
  - Nonfunctional requirements, which impose constraints on the design or implementation (such as performance requirements, quality standards or design constraints).

Requirements vs. Design

- Many people have difficulty understanding the difference between scope, requirements and design.
  - Scope demonstrates the needs of the organization, and is documented in a vision and scope document
  - Requirements document the behavior of the software that will satisfy those needs
  - Design shows how those requirements will be implemented technically

Change Control

- *Change control* is a method for implementing only those changes that are worth pursuing, and for preventing unnecessary or overly costly changes from derailing the project.
  - Change control is an agreement between the project team and the managers that are responsible for decision-making on the project to evaluate the impact of a change before implementing it.
  - Many changes that initially sound like good ideas will get thrown out once the true cost of the change is known.

Change Control

- A *change control board* (CCB) is made up of the decision-makers, project manager, stakeholder or user representatives, and selected team members.
  - The CCB analyzes the impact of all requested changes to the software and has the authority to approve or deny any change requests once development is underway.
  - Before the project begins, the list of CCB members should be written down and agreed upon, and each CCB member should understand why the change control process is needed and what their role will be in it.

Change Control

- Whenever a change is needed, the CCB follows the *change control process* to evaluate the change:
  - The potential benefit of the change is written down, and the project manager works with the team to estimate the potential impact that the change will have on the project.
  - If the benefit of the change is worth the cost, the project manager updates the plan to reflect the new estimates. Otherwise, the change is thrown out and the team continues with the original plan.
  - The CCB either accepts or rejects the change.