SE 463: Software Requirements Specification and Analysis

Michael Noukhovitch

Spring 2016, University of Waterloo

Notes written from Joanne Atlee's lectures.

Contents

1 Introduction

1.1 Why

Software specs and requirements are necessary to prevent future repair costs (req cost post release = 200*cost during requirements phase).

1.2 Common Problems

Requirements are:

- vague
- over-specified
- ambiguous
- changing
- incomplete
- infeasible
- contradictory

2 RE Reference Model

2.1 Objectives

Want to identify and articulate:

Requirements conditions and/or capabilities that describe a problem (must be achieved to get solution)

- desired changes to the world
- expressed in terms of environment

Specification complete, precise, verifiable description of the proposed system

- requirement re-expressed in terms of interface
- no constraints on design or implementation
- show that specs imply requirements through *assumptions*

2.2 Deriving Specs

For each requirement, *Req*, determine

- the specification, *Spec*, of how the system will monitor/control environment
- what domain knowledge assumptions, *Dom* are needed to link environmental constraints to system constraints (*e.g.* a plane is moving on the runway if its wheels are turning)

Mathematically, show that *at minimum* $Spec \wedge Dom \models Req$

3 Stakeholders

3.1 Scoping

purpose rationale for why the project is wanted

goal high level measurable criteria of success

scope business area affected by installation of system

stakeholders people with an interest in the system

constraints restrictions on scope or style of system

context diagram graphical model of context

- modularizes phenomena into domains
- system in center with arrows (actions) between it and stakeholders

3.2 Stakeholders

Stakeholder someone with stake in the ultimate success of the product (but sometimes **proxies** will do)

- owner/client: person paying for software
- customer: buys software after development
- users: experts on current system or similar
- domain experts: familiar with problem that software might solve
- software engineer: techonology expert
- inspectors
- market researchers
- lawyers
- industry standards
- experts on *adjacent* systems
- negative stakeholder: don't want you to succeed

4 Use Cases

4.1 Understanding The Work

- understand what the existing work is
  - don't think about implementation
  - abstract inputs/outputs
- study what you are permitted to change
  - what you need to know to decide a change
  - anything that can be affected by your product

4.2 Business Use Cases

Use case vertical slices of work made to reduce complexity

- represent end-to-end functionality
- triggered by external event
- captures complete response to event
- ideally orthogonal to one another

4.3 Use Case Models

actor generalization actor inheritance + polymorphism

■inlcude■ sub use case used within multiple other use cases

■extend■ sub use case that extends or replaces end of other use cases