CIS 422/522

Software Requirements
& a Little Quality Assurance

Requirements Phase Goals

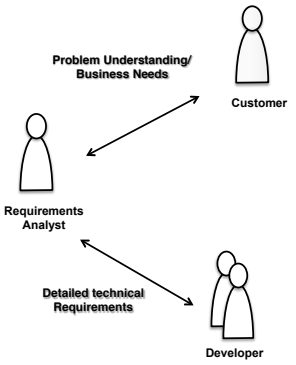
- What does "getting the requirements right" mean in the systems development context?
- Only three goals
  1. Understand precisely what is required of the software
  2. Communicate that understanding to all of the parties involved in the development (stakeholders)
  3. Control production to ensure the final system satisfies the requirements
- Sounds easy but hard to do in practice
- Understanding what makes these goals difficult to accomplish helps us understand how to mitigate the risks

A Requirements Process Framework

- Requirements Understanding
  - Requirements Elicitation - establish "what people want"
  - Requirements Negotiation - resolve stakeholder conflicts
- Requirements Specification
  - Concept of Operations - communicate with non-programming audiences
  - Software Requirements Specification - specify precisely what the software must do
- Requirements Validation and Verification
  - Establish that we have the right requirements (feedback)
  - Ensure our specification is good quality

Communicating with Different Audiences

- Customer/User
  - Focus on problem understanding
  - Use language of problem domain
  - Technical if problem space is technical
- Development organization
  - Focus on system/software solutions
  - Use language of solution space (software)
  - Precise and detailed enough to write code, test cases, etc.



Documentation Approaches

- **ConOps**
  - Informal requirements to describe the system's capabilities from the customer/user point of view
    - Purpose is to answer the questions, "What is the system for?" and "How will the user use it?"
    - Tells a story: "What does this system do for me?"
    - Focus on communication over rigor
- **SRS**
  - More formal, technical requirements for the development team
    - Purpose is to answer specific technical questions quickly and precisely
      - E.g. "What should the system output for this set of inputs?"
      - Reference, not a narrative, does not "tell a story"
    - Focus on precision and rigor
      - Goal is requirements that are precise, unambiguous, complete, and consistent

SRS Template

1. Introduction

1.1 Intended Audience and Purpose

<Describes the set of stakeholders and what each stakeholder is expected to use the document for. If some stakeholders are more important than others, describes the priorities.>

1.2 How to use the document

<Describes the document organization. This section should answer for the reader: "Where do I find particular information about X?">

Informal, user centric

2. Concept of Operations

<Use this section to give a detailed description of the system requirements from a user's point of view. The ConOps should be readable by any audience familiar with the application domain but not necessarily with software. The ConOps should make clear the context of the software and the capabilities the system will provide the user.>

2.1 System Context

<Specify the system boundaries including, particularly, the inputs and outputs. May include an illustration or context diagram. >

2.2 System capabilities

<System capabilities may be described in prose or with informal scenarios.>

Formal, technical

3. Behavioral Requirements

<Specification of the observable system behavior.>

3.1 System Inputs and Outputs

3.2 Detailed Output Behavior

<A black box specification of the visible, required behavior of the system outputs as a function of the system inputs. Tables, functions, use cases or other methods of specification may be used.>

ConOps: Informal Specification Techniques

- Use natural language and other informal methods
  - Use cases
  - Mock-ups (pictures)
  - Story boards
- Benefits
  - Requires little technical expertise to read/write
  - Useful for communicating with a broad audience
  - Useful for capturing intent (e.g., how does the planned system address customer needs, business goals?)
- Drawbacks
  - Inherently ambiguous, imprecise
  - Cannot effectively establish completeness, consistency
- However, can add rigor with standards, templates, etc.

Scenario Analysis Process

Applying scenario analysis in the requirements process

- Requirements Elicitation
  - Identify stakeholders who interact with the system
  - Collect "user stories" - how people would interact with the system to perform specific tasks
- Requirements Specification
  - Record as use-cases with standard format
  - Use templates to standardize, drive elicitation
- Requirements verification and validation
  - Review use-cases for consistency, completeness, user acceptance
  - Apply to support prototyping
  - Verify against code (e.g., use-case based testing)

Creating Use Cases

- Identify a key *actor* and *purpose*
  - The purpose informs the use case title and description
- Identify the main flow (ideal path) from the starting point to the result
  - Preconditions: anything that must be true to initiate the Use Case
  - Trigger: event, if any, initiating the Use Case
  - Basic Flow: sequence of interactions from the trigger event to the result
  - Alternative Flows: identify sequences branching off the Basic Flow

Guidelines for Good Use Cases

- Use Cases should express requirements, not design
  - Focus on import *results* that provide *value* to specific actors
    - I.e., if nobody really cares about the outcome, it is not a good use case
  - Focus on *what* the actor is doing, not the details of *how*
    - Not: "The user left-clicks on the radio button labeled *Balance* and presses the *Enter* button"
    - "The user elects the option to view the balance."
- Looking for a small number of use cases that capture the most important interactions

Use-Case Specification – Register for Courses

Brief Description

This use case allows a Student to register for course offerings in the current semester. The Student can also modify or delete course selections if changes are made within the add/drop period at the beginning of the semester. The Course Catalog System provides a list of all the course offerings for the current semester.

Actors

1. ***Primary Actor – Student***
2. ***Secondary Actor - Course Catalog System***

Flow of Events

1. ***Basic Flow***

1.1. LOG ON.
This use case starts when a student accesses the Course Registration System. The student enters a student ID and password and the system validates the student.

1.2. CREATE SCHEDULE.
The system displays the functions available to the student. These functions are: Create A Schedule, Modify a Schedule and Delete a Schedule. The student selects 'Create a Schedule'.

1.3. SELECT COURSES
The system retrieves a list of available course offerings from the Course Catalog System and displays the list to the student. The Student selects up to 4 primary course offerings and 2 alternate course offerings from the list of available offerings. The student can add and delete courses as desired until choosing to submit the schedule.

1.4. SUBMIT SCHEDULE.
The student indicates that the schedule is complete. The system validates the courses selected and displays the schedule to the student. The system displays the confirmation number for the schedule. The systems saves the student's schedule information. The use case ends.

Voting System Example

- Who are the actors
- What are the major tasks?
- What are the outcomes?
- What would be an alternative flow?

Technical Specification

The SRS

The role of rigorous specification

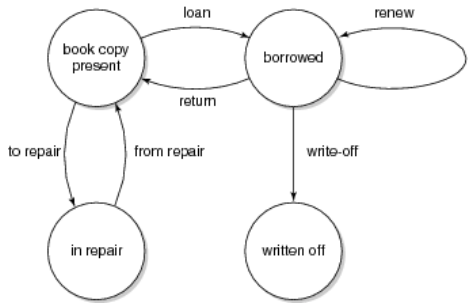
Requirements Documentation

- Is a detailed requirements specification necessary?
- How do we know what "correct" means?
  - How do we decide exactly what capabilities the modules should provide?
  - How do we know which test cases to write and how to interpret the results?
  - How do we know when we are done implementing?
  - How do we know if we've built what the customer asked for (may be distinct from "want" or "need")?
  - Etc…
- Correctness is a *relation* between a spec and an implementation (M. Young)
- Implication: until you have a spec, you have no standard for "correctness"

Technical Requirements

- Focus on developing a technical specification
  - Should be straight-forward to determine acceptable inputs and outputs
  - Preferably, can systematically check completeness consistency
- A little rigor in the right places can help a lot
  - Adding formality is not an all-or-none decision
  - Use it where it matters most to start (critical parts, potentially ambiguous parts)
  - Often easier, less time consuming than trying to say the same thing in prose
- E.g. in describing conditions or cases
  - Use predicates (i.e., basic Boolean expressions)
  - Use mathematical expressions
  - Use tables where possible

Example state transition diagram



SE, Modeling, Hans van Vliet, ©2008

Formal Specification Example

Type Dictionary

Name	Base Type	Units	Legal Values	Comment
Speed	Integer	Knots	[0, 250]	Speed measured in nautical miles per hour.
Weight	Integer	percent	[0,100]	Weighting for weighted average
time	Integer	seconds	time > 0	Time in seconds.

Monitored Variable Dictionary

Name	Type	Initial Value	Accuracy	Comment
LowResWS1	Speed	0	1	Wind speed reported by first low resolution sensor
LowResWS2	Speed	0	1	Wind speed reported by second low resolution sensor
HighResWS1	Speed	0	2.5	Wind speed reported by first high resolution sensor
HighResWS2	Speed	0	2.5	Wind speed reported by second high resolution sensor

Controlled Variable Dictionary

Name	Type	Initial Value	Accuracy	Comment
TransmWindSpeed	MsgType	ShortMsg	N/A	Transmitted value of wind speed

- SCR formal model
  - Define explicit types
  - Variables monitored or controlled

Quality Requirements

Terminology

- Avoid “functional” and non-functional" classification
- Behavioral Requirements – any information necessary to determine if the run-time behavior of a given implementation constitutes an acceptable system
  - All quantitative constraints on the system's run-time behavior
  - Other objective measures (safety, performance, fault-tolerance)
  - In theory all can be validated by observing the running system and measuring the results
- Developmental Quality Attributes - any constraints on the system's static construction
  - Maintainability, reusability, ease of change (mutability)
  - Measures of these qualities are necessarily relativistic (I.e., in comparison to something else

Behavioral vs. Developmental

Behavioral (observable)

- Performance
- Security
- Availability
- Reliability
- Usability

Properties resulting from the properties of components, connectors and interfaces that exist at run time.

Developmental Qualities

- Modifiability(ease of change)
- Portability
- Reusability
- Ease of integration
- Understandability
- Support concurrent development

Properties resulting from the properties components, connectors and interfaces that exist at design time *whether or not they have any distinct run-time manifestation.*

Specifying Quality Requirements

- Is it important to specify the quality requirements explicitly? Unambiguously?
  - Hint: what role would quality requirements play in customer acceptance?
- Are these kinds of specifications adequate?
  - "The system interface shall be easy to use."
  - "The system shall maximize the number of user transactions"

Specifying Quality Requirements

- When using natural language, write *objectively verifiable* requirements when possible
  - Maintainability: "The following kinds of requirement changes will require changes in no more than one module of the system…"
  - Performance:
    - "System output X has a deadline of 5 ms from the input event."
    - "System output Y must be updated at a frequency of no less than 20 ms."

Example Timing Requirements

5.2. TIMING REQUIREMENTS FOR DEMAND FUNCTIONS

For all the demand functions, the rate of demand is so low that it will not constitute a significant CPU-load.

For the starred entries, the desired maximum delay is not known; the entry is the maximum delay in the current OFP, which we will use as an approximation. In one case, both the current and desired values are given. The current value would be good enough to satisfy requirements, but the desired rate would be preferred.

Function name	Maximum delay to completion
IMS:	
Switch AUTOCAL light on/off	*200 ms
Switch computer control on/off	*200 ms
Issue computer failure	not significant
Change scale factor	*200 ms
Switch X slewing on/off	*200 ms
Switch Y slewing on/off	*200 ms
Switch Z slewing on/off	*200 ms
Change latitude-greater-than-70-degrees	*200 ms
Switch INA light on/off	*200 ms
FLR:	
Enable radar cursor	200 ms
Slave or release slave	40 ms

Requirements Validation and Verification

- Feedback-control for requirements
- Should answer two distinct questions:
  - Validation: "Are we building to the right requirements?"
  - Verification: "Are we building what we specified?"
  - The book is confused on the distinction
    - Checking internal consistency (agreement with itself) is verification
    - Checking external consistency (agreement with the world) is validation
- Validation requires going back to the stakeholders: can use many techniques
  - Review of specifications
  - Prototyping
  - Story-boarding
  - Use case walkthroughs
  - Review software iterations
- Verification requires checking work products against specifications
  - Review
  - Testing
  - Formal modeling and analysis

Questions?

Assignments

- Set up instructor meetings this week
- Finish incomplete drafts of deliverables