Managing Software Requirements in DSD

Stuart Faulk
University of Oregon

Outline

- Lecture: Software Requirements
  - Importance of requirements (real development context)
  - Project case study: the Floating Weather Station (FWS)
  - Requirements risks in DSD
- Lab: analyzing likely requirements changes
- Lecture: communicating requirements
  - Informal specification
  - Formal specification
- Lab: negotiate and specify new wave height requirements

What is a “software requirement?”

- A description of something the software must do or property it must have
- The set of system requirements denote the problem to be solved and any constraints on the solution
  - Ideally, requirements specify precisely what the software must do without describing how to do it
  - Any system that meets requirements should be an acceptable implementation
1. The majority of software errors are introduced early in software development.

2. The later that software errors are detected, the more costly they are to correct.

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 straightforward but hard to do in practice
- Understanding what makes these goals difficult to accomplish helps us understand how to mitigate the inherent risks

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements...No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later."

F.P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering"
What makes requirements difficult?

- Comprehension (understanding)
  - People don’t (really) know what they want (…until they see it)
  - Superficial grasp is insufficient to build correct software

- Communication
  - People work best with regular structures, conceptual coherence, and visualization
  - Software’s conceptual structures are complex, arbitrary, and difficult to visualize

- Control (predictability, manageability)
  - Difficult to predict which requirements will be hard to meet

- Inseparable Concerns
  - Must make tradeoffs where requirements conflict: requires negotiation
  - Many requirements issues cannot be cleanly separated (i.e., decisions about one necessarily impact another)

Additional Risks of DSD

- DSD tends to aggravate existing difficulties
- Comprehension
  - Different languages, cultures, expectations
  - Greater risk of ambiguity, misunderstanding
- Communication
  - Reduced communication: limited bandwidth, language, culture, invisible stakeholders
  - More difficult to negotiate common understanding, may miss problems
- Control
  - Less feedback, often delayed
  - Easy to get out of sync and not know it
- Inseparable concerns
  - Difficulty making clean divisions, allocation of responsibility among sites
  - Conversely, easy to have inadvertent side effects on other code
Risk Mitigation Strategies

- Build risk mitigation into the project’s software process
  - Requirements changes occur throughout development
  - Must address at all stages of process
- Requirements are missing, misunderstood
  - Requirements exploration with stakeholders (customer)
  - Early modelling: prototypes, mockups
  - Careful review
  - Incremental delivery
  - Clear responsibilities for requirements tasks, products
- Requirements change
  - Consider the effects of changes in advance
  - Software design for robustness, ease of change
  - Explicit processes for managing change

Requirements Process Components

- Process view: activities, artifacts, roles, and relations
- Activities: deploy explicit requirements activities
  - Requirements exploration and understanding
  - Requirements negotiation (and explicit signing)
  - Requirements specification
  - Requirements verification and validation (feedback)
  - Change management for distributed team
- Artifacts: provide vehicles for capture, communication & assessment, e.g.,
  - Prototypes, mock-up, story board, use cases
  - Common requirements specification
  - Reviews
- Roles: create clear responsibilities for activities, artifacts, and communication, e.g.,
  - Analyst: exploration, customer interaction, negotiation
  - Requirements Engineer: specification
  - Reviewer: verification and validation

Process View
Requirements Elicitation & Negotiation

- Focuses on the question: “Are we building the right system?”
- Process activities
  - Requirements analysis: interact with customer and other stakeholders
    - Identify stakeholders
    - Identify both functional and “non-functional” requirements (performance, maintainability, etc.)
    - Identify likely changes (evolution)
  - Negotiate delivered capabilities with stakeholders
  - Find consensus among distributed teams
  - Explicit agreement on requirements (signoff)
- Products
  - Anything useful to ensuring a correct understanding of the problem domain and stakeholder desires
  - Examples: mock ups, use cases, demonstration prototypes

Class Case Study
The Floating Weather Station

A Floating Weather Station

Floating weather stations (FWS) are buoys that float at sea and are equipped with sensors to monitor weather conditions. Each FWS has an on-board computer that maintains a history of recent weather data. At regular intervals, the buoy transmits the weather data using a radio transmitter.
Informal Requirements

Initial FWS Requirements

Floating weather stations (FWS) are buoys that float at sea and that are equipped with sensors to monitor weather conditions. Each FWS has an on-board computer that maintains a history of recent weather data. At regular intervals the buoy transmits the weather data using a radio transmitter.

The initial prototype for the buoy will measure the wind speed in knots. The buoys will use four small wind speed sensors (anemometers): two high-resolution sensors and two, less expensive, low-resolution sensors.

Accuracy is software enhanced by computing a weighted-average of the sensor readings over time. Each sensor is read once every second with the readings averaged over four readings before being transmitted. The calculated wind speed is transmitted every two seconds.

Over the course of development and in coming versions, we anticipate that the hardware and software will be routinely upgraded including adding additional types of sensors (e.g. wave height, water temperature, wind direction, air temperature). A system that can be rapidly revised to accommodate new features is required.

Expected changes

- The formula used for computing wind speed from the sensor readings may change.
- In particular, the weights used for the high resolution and low-resolution sensors may change.
- Device
- The wind speed sensor hardware on a FWS may change.

Use Case

Use Case

Each two seconds the buoy transmits the value of the wind speed in knots. The weighted average is computed over the current and three previous readings of all the on-board sensors.

Change Case

The customer determines a more accurate algorithm for computing the wind speed.

A change in sensor hardware requires changing the weightings given to different types of wind speed sensors.
**Requirements Changes**

- Assertion: for DSD it is important to think ahead about how the requirements are likely to change during development and maintenance, and document these in the requirements specification.
- Why? What can happen as requirements change?

**Lab Exercise: FWS Likely Changes**

- Assume that it is important to think ahead about how the requirements are likely to change over the life of the system.
  - We will talk about how this information is used later.
- Work with your team to identify the kinds of things that are likely to change for the FWS.
- Create a wiki page to record the likely changes.
  - Think of them as “change scenarios.”
  - Later, we will evaluate the design using these scenarios: i.e., “How must the design change if requirements change C happens?”
Initial FWS Requirements

Floating weather stations (FWS) are buoys that float at sea and that are equipped with sensors to monitor weather conditions. Each FWS has an on-board computer that maintains a history of recent weather data. At regular intervals the buoy transmits the weather data using a radio transmitter.

The initial prototype for the buoy will measure the wind speed in knots. The buoys will use four small wind speed sensors (anemometers): two high-resolution sensors and two, less expensive, low-resolution sensors.

Accuracy is software enhanced by computing a weighted-average of the sensor readings over time. Each sensor is read once every second with the readings averaged over four readings before being transmitted. The calculated wind speed is transmitted every two seconds.

Over the course of development and in coming versions, we anticipate that the hardware and software will be routinely upgraded including adding additional types of sensors (e.g. wave height, water temperature). A system that can be rapidly revised to accommodate new features is required.

Expected changes

- C1. The formula used for computing wind speed from the sensor readings may change. In particular, the weights used for the high resolution and low-resolution sensors may change.
- C8. The wind speed sensor hardware on a FWS may change.

Likely Kinds of Changes

- Think about the individual devices and requirements
- May also want to prioritize these

Behavior
- C1. The formula used for computing wind speed from the sensor readings may change. In particular, the weights used for the high resolution and low-resolution sensors may change.
- C2. The format of the messages that an FWS sends may change.
- C3. The transmission period of messages from the FWS may change.
- C4. The rate at which sensors are scanned (sensor period) may change.
- C5. The number of readings of each sensor used (the history of the sensor) may change.

Devices
- C6. The number and types of wind speed sensors on a FWS may change.
- C7. The resolution of the wind speed sensors may change.
- C8. The wind speed sensor hardware on a FWS may change.
- C9. The transmitter hardware on a FWS may change.
- C10. Add new kinds of sensors and associated outputs.

Likely Kinds of Change

- Examples/FWS_ProseRequirements v2.pdf
Lecture: Requirements in DSD

- Stakeholder and requirements communication
- Need for documentation
- Informal specification
- Formal specification

Additional Risks of DSD

- DSD tends to aggravate existing difficulties
- Comprehension
  - Different languages, cultures, expectations
  - Greater risk of ambiguity, misunderstanding
- Communication
  - Reduced communication: limited bandwidth, language, culture, invisible stakeholders
  - More difficult to negotiate common understanding, may miss problems
- Control
  - Less feedback, often delayed
  - Easy to get out of synch and not know it
- Inseparable concerns
  - Difficulty making clean divisions, allocation of responsibility among sites
  - Conversely, easy to have inadvertent side effects on other code

Risk Mitigation Strategies

- Build risk mitigation into the project's software process
  - Requirements changes occur throughout development
  - Must address at all stages of process
- Requirements are missing, misunderstood
  - Requirements exploration with stakeholders (customer)
  - Early modeling: prototypes, mockups
  - Careful review
  - Incremental delivery
  - Clear responsibilities for requirements tasks, products
- Requirements change
  - Consider the effects of changes in advance
  - Software design for robustness, ease of change
  - Explicit processes for managing change
Real DSD => Diverse Stakeholders

Requirements Communication

- Focus on the question: “Do all of the stakeholders share a common understanding of what must be built?”
  - Particularly critical for DSD
- Process activities
  - Requirements specification: capture complete requirements in sufficient detail to serve the needs of all of the stakeholders
  - Requirements verification: check that the requirements specification is of adequate quality (complete, consistent, testable, etc.)
  - Requirements communication: check for consistent understanding
    - Usually implicit in co-located projects
    - Should be incorporated as part of verification (e.g., distributed reviews, validation)
- Products
  - Requirements documentation
  - Reviews

Purposes and Stakeholders

- Many potential stakeholders using requirements for different purposes
  - Customers: the requirements typically document what should be delivered and may provide the contractual basis for the development
  - Managers: provides a basis for scheduling and a yardstick for measuring progress
  - Software Designers: provides the “design-to” specification
  - Coders: defines the range of acceptable implementations and is the final authority on the outputs that must be produced
  - Quality Assurance: basis for validation, test planning, and verification
  - Also: potentially Marketing, regulatory agencies, etc.
- For DSD, must specifically consider needs of all distributed groups
Needs of 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)
  - Practice and detailed enough to write code, test cases, etc.

Two Kinds of Software Requirements

- Communicate with customers: i.e., stakeholders who understand the problem domain but not necessarily programming (solution domain)
  - Do not understand computer languages but may understand technical domain-specific languages
  - Must develop understanding in common languages
- Communicate with developers: sufficiently precise and detailed to code-to, test-to, etc.
  - Stated in the developer’s terminology
  - Addresses properties like completeness, consistency, precision, lack of ambiguity
- For businesses, these may be two separate documents

Documentation Approaches

- 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?”
  - Helps to use a standard template
- More formal, technical requirements for development team (architect, coders, testers, etc.)
  - Purpose is to answer specific technical questions about the requirements quickly and precisely
  - Answers, “What should the system output in this circumstance?”
  - Reference, not a narrative, does not “tell a story”
  - Goal is to develop requirements that are precise, unambiguous, complete, and consistent
  - What are the problems with use cases for this purpose?
Informal Techniques

- Most requirements specification methods are informal
  - Natural language specification
  - 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.

Mock-up Example

Can use simple technology
Emailed
Commented on
Reviewed

Mock-up Example

A systematic approach to use cases
- Uses a standard template
- Easier to check, read
- Still informal
Technical Specification

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”

From Example SRS/SDS/Plan

SOFTWARE REQUIREMENT SPECIFICATION .......................... 1
PURPOSE OF THE DOCUMENT ................................................. 1
PROBLEM STATEMENT ......................................................... 1
PROPOSED SOLUTION .......................................................... 1
USER CHARACTERISTICS ....................................................... 2
USER SCENARIOS ................................................................. 3
FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS ............... 4
OTHER REQUIREMENTS ......................................................... 5
Technical Requirements

- Focus on developing a technical specification
  - Should be straight-forward to determine acceptable inputs and outputs
  - Preferably, can systematically check completeness and 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

Formal Specification Example

<table>
<thead>
<tr>
<th>Type Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Speed</td>
</tr>
<tr>
<td>Weight</td>
</tr>
<tr>
<td>Time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Monitored Variable Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>LowRes/WindSpeed</td>
</tr>
<tr>
<td>LowRes/WindDirection</td>
</tr>
<tr>
<td>HighRes/WindSpeed</td>
</tr>
<tr>
<td>HighRes/WindDirection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Controlled Variable Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Transmit/WindSpeed</td>
</tr>
</tbody>
</table>

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

Formal Specification Example

**Transmit/WindSpeed Event Function**

The transmitted wind speed is a moving, weighted average over the length of the history of sensor readings defined as follows:

LWS = LowRes/Wind, HWS=HighRes/Wind, H = History

For i be the current index of all sensor readings and \( e[i] \) be the \( i^{th} \) sequential value of variable \( v \) (hence LWS[i] in the most recent value of LWS).
Control Production

- Control production to ensure the final system satisfies the requirements: this means:
  - Team members use the requirements specification as the basis of development
    - Designing the architecture: requirements define all external design goals, e.g.: expected changes, subsets, extensions, etc.
    - Designing modules to provide required capabilities
  - Team members use the requirements spec as a basis for verification
    - Verify designs against requirements
    - Basis of system test planning (as opposed to module)
  - If the spec is not adequate for these purposes, then fix the spec!

Summary

- Requirements characterize “correct” system behavior
- Being in control of development requires:
  - Getting the right requirements
  - Communicating them to the stakeholders
  - Using them to guide development
- Requirements activities must be incorporated in the project plan
  - Requirements baseline
  - Requirements change management

Questions?
Exercise

- Work with your team to agree on requirements specification for new FWS capabilities
- Add requirements to your *assemble* workspace

New FWS Requirements

Management has decided to extend the prototype to include two new sensors for monitoring wave height. Wave height is defined as the distance from the water's lowest point (the trough) to its highest point (the peak).

**System Input:**

There are two sensors each of which is most accurate in certain sea conditions. The sensors output their values every 0.1s:

1. The absolute Wave Height sensor produces a non-negative integer value that gives the current wave height in centimeters to an accuracy of 1 cm.
2. The incremental Wave Height sensor produces an integer value (which may be negative) that represents the change in centimeters of water level from a central point (i.e., sea level) also to an accuracy of 1 cm.

**System Output:**

Choose some value related to the wave height to output. Possible examples include:

- Maximum wave height: the maximum height of the waves over some number of readings or some height of time.
- Average wave height: the average height of the waves over some number of readings.
- Minimum wave height
- Rate of change of wave heights
- Others?

**Exercise:**

Your job is to agree on and provide a set of requirements for your prototype as you work with your team. Output of the exercise should include a written specification of the requirements on your team wiki. The requirements should include:

- Required output (transmitted) values
- Any additional likely changes implied by the new requirements.

Example

Informal requirements for a wave height maximum

**System Input:**

The system shall transmit the maximum value of the wave height detected by either sensor over a history of sensor readings to an accuracy of 1 cm.

Initially, in this case, if $H(1)$ for the 1st value of the incremental wave height sensor and $AH(1)$ for the 1st value of the absolute wave height sensor, then the minimum wave height (MWH) should be given by $AH(1) - MWH = \max(AH(1), MWH) - AHN(1)$.

for i = 1, 2, 3, ... where $i$ is the most recent sensor reading and $n$ is the 1st preceding reading.

**System Output:**

Behavior:

C1: The formula used for determining the maximum wave height might change (e.g., by weighting the sensors, etc.).

C2: The number of sensor readings used might change.

Environment:

C3: The number and type of wave height sensors might change.

C4: The resolution of the wave height sensors might change.
Questions?