Managing Software Requirements in DSD

Outline

- A few comments
- Importance of requirements (real development context)
- Requirements goals
- What makes the goals difficult to achieve?
  - What additional difficulties are introduced by DSD?
- Requirements understanding
- Requirements communication
  - Why are written requirements necessary?
- Requirements Verification

© S. Faulk 2010
Increment 2

- Requirements documentation
  - What new functionality will be provided? (Definitely, probably, maybe)
  - What improvements to existing functionality and/or quality will be provided?
- Extended architectural design
  - What will change in the architectural design?
  - What will be added, refactored, etc?
- Plan
  - What are the intermediate milestones?
  - Who will be responsible for what?
  - How will the milestones be objectively checked to determine progress against the plan?
  - What are the biggest risks? How will you mitigate them?
  - How has the plan been adjusted to avoid or mitigate the main problems in the first increment?

On Planning

No battle project plan ever survives contact with the enemy software.
- (apologies to) H. von Moltke

- Does not mean you don’t need a plan
- Rather plan should include contingencies (“what is our fallback plan if X happens”?)

© S. Faulk 2010
Teamwork Comments

- Criticism and sensitivity
  - Be able to give and take constructive criticism
  - Best possible source for improvement
  - Focus on the artifact and objective properties (not the person)
    - i.e., “the module fails on these test cases,” not “you forgot these cases”
  - This is easiest for everyone if review is the norm (standard practice for every artifact)
  - Goal: develop a culture of review

- Responsibility and trust
  - Do the work you commit to
  - Trust teammates to do theirs
  - Particularly an issue for distributed teams since trust is hard to establish or maintain
  - “Taking over” someone else’s work tends to destroy trust, review instead

Software 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

Importance of Getting Requirements Right

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

© S. Faulk 2010
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 straight-forward 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.”

Inherent Difficulties

- 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
  - Requirements change all the time
  - Together can make planning unreliable, cost and schedule unpredictable
- Inseparable Concerns
  - Many requirements issues cannot be cleanly separated (i.e., decisions about one necessarily impact another)
    - E.g., performance vs. security, safety vs. time to market
  - Difficult to apply “divide and conquer”
  - Must make tradeoffs where requirements conflict: requires negotiation among stakeholders with competing interests

Additional Risks of DSD

- DSD tends to aggravate existing difficulties (as we have experienced)
- Comprehension
  - Different languages, cultures, expectations
  - Greater risk of ambiguity
- 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
Requirements Engineering Strategies

- Apply risk mitigation in the context of the project's software process and derived plan
- Activities: deploy explicit requirements activities
  - Requirements exploration and understanding
  - Requirements negotiation (and explicit signoff)
  - Requirements specification
  - Requirements verification and validation (feedback)
- Artifacts: provide vehicles for capture, communication & assessment, e.g.,
  - Prototypes
  - 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

---

Suggested Approach

- Plan process activities and artifacts around the three goals
  1. Understand precisely what is required of the software
     - Requirements elicitation, analysis: interact with customer and other stakeholders
     - Develop documentation and other materials
  2. Communicate that understanding to all of the parties involved in the development (stakeholders)
     - Develop documentation and other materials
     - Incorporate feedback
  3. Control production to ensure the final system satisfies the requirements
     - Verification of architecture, module specs, etc. against requirements
- For each of these goals, need to ask the fundamental questions
  - How do we address this goal in the process?
  - How do we check how well we are meeting the goal?
1. Requirements Understanding

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.)
  - Negotiate delivered capabilities with stakeholders
- Requirements validation: implement feedback loops
- Requirements changes: processes for addressing requirements changes

Products
- Anything useful to ensuring a correct understanding of the problem domain and stakeholder desires
- Examples: mock ups, use cases, demonstration prototypes

Real DSD => Diverse Stakeholders
2. 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 spec 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
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”

© S. Faulk 2010

19

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

*this was partially given for the first increment

© S. Faulk 2010

20
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)
  - Precise 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
- For our purposes, can be separate sections in one physical document
Documentation Approaches

- Informal requirements to describe the system's capabilities from the customer/user point of view
  - E.g., Use cases, story boards,
  - 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?

From Example SRS/SDS/Plan

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

SRS or Technical Reference

Informal Operational Requirements

© S. Faulk 2010
1. Use Case: Manage Reports

1.1 Description
This Use Case describes operations for Creating, Saving, Deleting, Printing, Exiting and Displaying reports.

1.2 Actors
User
Project database

1.3 Triggers
Program Manager selects operations from menu.

1.4 Flow of events

1.4.1 Basic Flow
1. User chooses desired report by selecting “Report” -> “Open” from the menu bar
2. System displays report to screen
3. User selects desired report layout using Use Case Specify Report
4. Steps 2 and 3 are repeated until user is satisfied
5. User can Save or Print report using use case Save Report or Print Report
6. User Exits report by selecting “Exit” from the “File” menu

1.4.2 Alternative Flows
1.4.2.1 Create New Report
1. User selects “Create New Report” from file menu
2. ...

1.4.2.2 Delete Report
...

1.4.3 Preconditions

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

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

© S. Faulk 2010
A Real Requirement

(2.16.3.f) While acting as the bus controller, the C&C MDM CSCI shall set the e,c,w indicator identified in Table 3.2.16-II for the corresponding RT to "failed" and set the failure status to "failed" for all RT’s on the bus upon detection of transaction errors of selected messages to RT’s whose 1553 FDIR is not inhibited in the two consecutive processing frames within 100 millisecond of detection of the second transaction error if; a backup BC is available, the BC has been switched in the last 20 sec., the SPD card reset capability is inhibited, or the SPD card has been reset in the last 10 major (10-second) frames, and either:

1. The transaction errors are from multiple RT’s, the current channel has been reset within the last major frame, or
2. The transaction errors are from multiple RT’s, the bus channel’s reset capability is inhibited, and the current channel has not been reset within the last major frame.

Example of a NASA requirement for the Command and Control bus for the Space Station specifying exactly when all remote terminals on the bus should be switched to their backups

Q: Should the remote terminals be switched to their backups if the C&C MDM is acting as bus controller, there are transaction errors in two consecutive frames, errors are on selected messages, the RT’s 1553 FDIR is not inhibited, a backup BC is available, the BC has been switched on in the last 20 seconds, transaction errors are from multiple RT’s and the current channel has been reset in the last major frame?

Simple Home Heating Example

Hot water provides the system’s heat source. When heat is needed, the HHS turns on a furnace to heat water to 80 degrees Fahrenheit. When the water reaches minimum operating temperature (initially 80° F), the system must turn on the circulating pump causing hot water to flow to the radiators in the house. When heat is no longer needed, the system turns off the pump and shuts down the furnace.

Communicates precisely and concisely

© S. Faulk 2010

27 28
3. 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
Peer Reviews

- Peer Review: a process by which a software product is examined by peers of the product's authors with the goal of finding defects
- Why do we do peer reviews?
  - Review is often the only available verification method before code exists
  - Formal peer reviews (inspections) instill some discipline in the review process
- Particularly important for distributed teams
  - Supports communication and visibility
  - Provides feedback on both quality and understanding
    - i.e., makes the communication effectiveness and level of understanding visible
  - A good review shows communication is working!
Effectiveness of Peer Reviews

- Generally considered most effective manual technique for detecting defects
  - E.g., Bell-Northern found 1 hour code inspecting saves 2 to 4 hours code testing
  - Effect is magnified in requirements inspections (e.g., 30 times in one study)

- Means that you should be doing peer reviews, but...
  - Doesn’t mean that manual inspections cannot be improved
  - Doesn’t mean that manual inspections are the best way to check for every properties (e.g., completeness)
    - Should be one component of the overall V&V process
Peer Review Issues

- Several inefficiencies, particularly for large or distributed projects
- Tendency for reviews to be incomplete and shallow
  - Reviewers typically swamped with information, much of it irrelevant to the review purpose
  - Reviewers lack clear individual responsibility
- Effectiveness depends on reviewers to initiate actions
  - Review process requires reviewers to speak out
  - Keeping quiet gives lowest personal risk
  - Rewards of finding errors are unclear at best
- Process depends on group meetings
  - High overhead
  - Difficult to do for DSD
- No way to cross-check unstated assumptions

Desired Qualities of Effective Review

- Ensures adequate coverage of artifact in breadth and depth
- Reviewers review only issues on which they have expertise
- Review process is active: i.e., performing the review produces visible output (risk in in doing nothing)
- Individual responsibilities are clear and fulfilling them is evidence of a job well done
- Review process focuses on finding specific kinds of errors
- Small group communication in place of large meetings
  - Permit detailed discussion of issues
  - Expose where assumptions differ
Active Review Process

- **Role:** Quality Assurance

- **Activities**
  - Identify several types of review each targeting a different type of error (e.g., UI behavior, consistency between safety assertions and functions)
  - Identify appropriate classes of reviewers for each type of review (specialists, potential users, methodology experts)
  - Assign reviews to achieve coverage: each applicable type of review is applied to each part of the specification
  - Design review questionnaires (key difference)
  - Conduct review
  - Revise

- **Artifacts**
  - Input: documentation to be reviewed and support docs (e.g., module spec plus requirements/use cases)
  - Output: Review questionnaires

© S. Faulk 2010

Examples

- In practice: an active review asks a qualified reviewer to check a specific part of a work product for specific kinds of defects by answering specific questions, e.g.,
  - Ask a designer to check the functional completeness by showing the calls sequences sufficient to implement a set of use cases
  - Ask a systems analyst to check the ability to create required subsets by showing which modules would use which
  - As a developer to check the data validity of a module’s specification by showing what the output would be for in-range and out-of-range values
  - Ask a technical writer to check the SRS for grammatical errors
- Can be applied to any kind of artifact from requirements to code as long as the reviewer is forced to use the document

© S. Faulk 2010
Conventional vs. Active Questions

- Goal: Make the reviewer(s) think hard about what they are reviewing*
  - Define questions that the review must answer by using the specification
  - Target questions to bring out key issues
  - Phrase questions to require "active" answers (not just "yes")

<table>
<thead>
<tr>
<th>Conventional Design Review Questions</th>
<th>Active Design Review Questions*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Are exceptions defined for every program?</td>
<td>For each access program in the module, what exceptions that can occur?</td>
</tr>
<tr>
<td>Are the right exceptions defined for every program?</td>
<td>What is the range or set of legal values?</td>
</tr>
<tr>
<td>Are the data types defined?</td>
<td>For each data type, what are: • an expression for a literal value of that data type; • a declaration statement to declare a variable for that type; • the greatest and least values in the range of that data type?</td>
</tr>
<tr>
<td>Are the programs sufficient?</td>
<td>Write a short pseudo-code program that uses the design to accomplish (some defined task).</td>
</tr>
</tbody>
</table>

Individual Review Process

- Role: [Specialist] Reviewer
  - Where [Specialist] denotes area expertise like problem domain requirements, architectural design, module design, etc.

- Artifacts
  - Input: artifact under review (e.g., SRS, module spec), prepared review questions
  - Output: answers to questions, defects, issues

- Activities
  - Overview of artifact
  - Individual review – use artifact to answer questions
  - Discussion of results
    - One-on-one or small group, for DSD can be on-line
    - Focus on discussion of issues identified in review
    - Purpose of discussion is understanding of the issue (not necessarily agreement)
Role of Use Cases

- Use cases or scenarios can be effectively used in active review
- Apply requirements scenarios to verify design against requirements
  - “Show the sequence of program calls that would implement use case C”
  - “Which modules would have to change to add feature F (a likely change)?”
- Conversely, can check properties ask the reviewer to construct scenarios
  - “What sequence of calls would result in an exception E?”

Why Active Reviews Work

- Focuses reviewer’s skills and energies where they have skills and where those skills are needed
  - Questionnaire allows reviewers to concentrate on one concern at a time
  - No one wastes time on parts of the document where there is little possibility of return.
- Largest part of review process (filling out questionnaires) is conducted independently and in parallel
- Reviewers must participate actively but need not risk speaking out in large meetings
- Downside: much more work for V&V (but can be productively pursued in parallel with document creation)
Simple Integer Stack

//Module Interface Spec
void push(int newItem); //push an integer on the stack
int pop(); //Remove the top int from the stack
boolean isEmpty(); //Returns true if the stack is empty

What kinds of questions would one ask?
Suppose we are handling order numbers LIFO and different subsystems handle different groups of order numbers in parallel?

Summary

- Need to do reviews to find defects
- Critical for distributed teams
  - Provides another communication pathway
  - Makes level of understanding visible
- Active reviews are more efficient and effective but may take more effort
- Assignment
  - Choose one artifact (e.g., a module spec)
  - Review within your own team
  - Request a review from PKU side (I'll ask them to do the same)
Review Questions For Abstract Interfaces:

**Requirements Validity**

- For each service provided by the module, is the service valid for all expected uses of this module? If not, give an example of a use where the service is not valid.

- For each service provided by the module, is the service valid for all expected versions of this module? If not, give an example of a needed configuration or version where the service is not valid.

- For each service needed described in this specification, is a module (or set of modules) identified that this module is allowed to use to satisfy the need?

Review Questions For Abstract Interfaces:

**Requirements Sufficiency**

- Does the set of services provided specify all of the services that will be needed by users of this module? Are there any services defined that are not identified in the requirements?

- Does the set of services needed specify all of the services that this module will need from other modules in order to operate correctly? What services are needed that are not identified in the requirements?
Review Questions For Abstract Interfaces:

**Consistency Between Services Provided and Access Programs**

- For each Services Provided described in this specification, which access program(s) can be used to satisfy the service?

- For each access program and signal specified in sections 2 and 6 which Service Provided is satisfied by the access programs?

---

Review Questions For Abstract Interfaces:

**Access Program Adequacy**

- Is the set of access programs and signals sufficient to satisfy the uses needs of modules that are allowed to use this module?

- Are there access programs that should be combined into one access program?

- Are there single access programs that should be refactored into several different access programs?

- Are the performance requirements adequate for the uses that will be made of this module?
Questions?