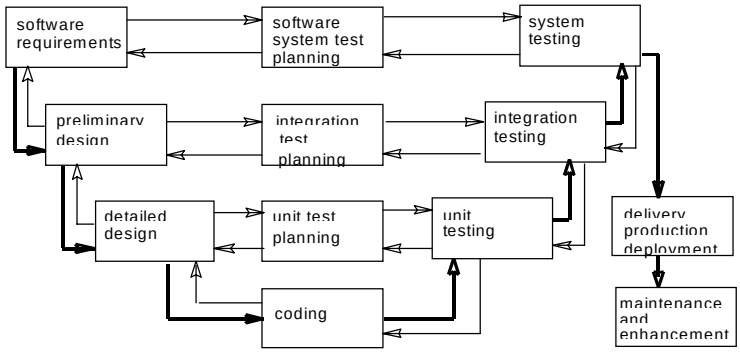
CIS 422:
Software Requirements

Stuart Faulk

Overview

- What is a “requirement?”
- Why do we care about requirements?
- What do we need to accomplish in the requirements phase?
  - Who is the audience for the SRS?
  - What do they use it for?
- What’s difficult about what we’re trying to do?
  - Essential difficulties
  - Accidental difficulties
- What goes in an SRS?
- What properties do we need in an SRS to do the job well?
- What is the role of formal methods?

Requirements in Software Life Cycle



A Software Life Cycle

Requirements Goals

- What are the goals of the requirements phase?

- Standard definition: “Establish and specify precisely what the software must do without describing how to do it.”
- Establish what to build be for starting to build it.
  - Make the “what decisions” explicitly before starting design … not implicitly during design.
  - Make sure you build the system that’s actually wanted/needed.
  - Allow the users a chance to comment before it’s built.
- Provide an organized reference document - the Software Requirements Specification (SRS)
  - Communicate the results of analysis
  - Provide a baseline reference document for developers and QA

Requirements Goals

- Why the emphasis on “what not how”?
  - Seek to describe the problem to be solved not the solution to ensure real problem is solved
  - Avoid over-constraining the design or implementation
  - Solutions are typically more complex, harder to understand, harder to change

Parts of the Requirements Phase

- Problem Analysis
  - The (“establish” part) also called “problem understanding, requirements exploration, etc
  - Goal is to understand precisely what problem is to be solved
  - Includes: Identify exact system purpose, who will use it, the constraints on acceptable solutions, and possible tradeoffs among conflicting constraints.
- Requirements Specification
  - Goal is to develop a technical specification - the Software Requirements Specification (SRS)
  - SRS specifies exactly what is to be built, captures results of problem analysis, and characterizes the set of acceptable solutions to the problem.

Do we Really Need Requirements?

- Do we always need a requirements specification?

- Focus large, complex systems
  - Multi-person: many developers, many stakeholders
  - Multi-version: intentional and unintentional evolution
- Qualitatively distinct from small developments
  - Multi-person introduces need for organizational functions (management, accounting, marketing), policies, oversight, etc.
  - More stakeholders and more kinds of stakeholders
- Quantitatively distinct from small developments
  - e.g., complexity of communication rises exponentially
  - Techniques that work in the small fail utterly (e.g., word of mouth)
- Rule of thumb: project starts to be "large" when group developing a single work product can't fit around a table.

Stakeholders

- For "large" developments, who has an interest in the content of the requirements specification (called stakeholders) and why?
- **Customers**: the requirements typically document what should be delivered and my 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.

Terminology

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

Sommerville’s “Requirements”

- What is the system required to do if an external file type has two or more associated tools?
- Is the user allowed to associate a tool with a file type or does the system do this from some other source? Change the association?
- Does the system have to associate a specific icon with each file?
- What happens if the user associates the same icon with two different file types?

Why are Requirements Hard to do Well?

"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"

A Big Problem

- Requirements issues appeared along with large software

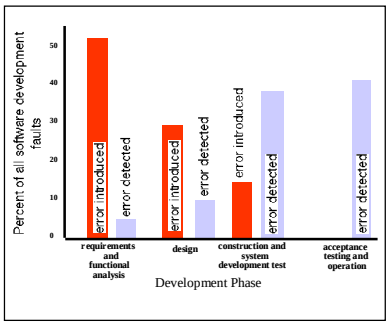
  Ballistic Missile Defense system developers note: “In nearly every software project that failed to meet performance and cost goals, requirements inadequacies played a major and expensive role in project failure” [Alford 79]

- Survey of major Aerospace firms turns up requirements as the top-priority problem.
- Recent GAO report identifies requirements as a major problem source in two-thirds of systems examined

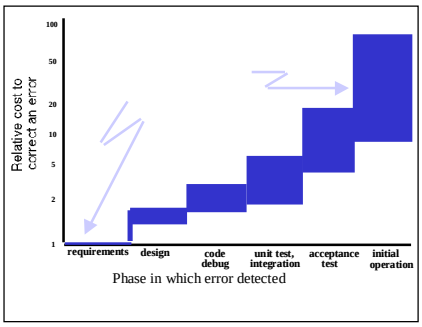
=> SE improvements haven’t kept pace with growth in project scale and complexity

Distribution and Effects of Requirements Errors

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.

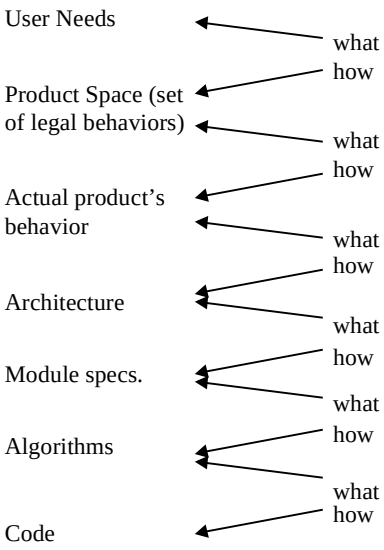


Requirements Goal

- Standard definition:
  "Identify and specify precisely what the software must do without describing how to do it."
  - Idiomatically: Specify "what" without specifying "how."

What's so hard about that?

What vs. How Dilemma



From Davis, Software Requirements

What vs. How (2)

- One man's "what" is another's "how"
- Typically mixed together
  - "If the system cannot complete the transaction within 1 second, the data base must be rolled back to it's initial state and the customer notified"
  - "Update data associated with each aircraft being tracked by the low aperture radar must be contained in distinct packets on the primary data bus."
- Most systems contain more than one problem space (e.g., user interface requirements vs. accounting transaction requirements).
- Upshot: Must agree on what constitutes the problem space and what constitutes the solution space (before the discussion even makes sense)
  - Requirements analysis and specification then belong in the problem space
  - Future issues: Does your approach help obtain and maintain this distinction?

Requirements Phase Goals

- Only three goals (given the large development context)
  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
- Problems in requirements result from the failure to adequately accomplish one of these goals
- What makes these goals difficult to accomplish?

Essential vs. Accidental

- Thesis:

  Meeting the requirements phase goals is
  essentially difficult
  - but not as difficult as we make it!

- Essential difficulties - part of the essence of the problem
- Accidental difficulties - difficulties introduced or added by the way we do things

*(not “essential” as in “needed” nor “accidental” as in “unintended”)

Essential Difficulties

- **Comprehension** (understanding)
  - People don’t (really) know what they want (… until they see it)
  - Superficial grasp is insufficient for expressing requirements

Essential Difficulties

- **Communication**
  - People work (think) best with regular structures, conceptual coherence, and visualization
  - Software's conceptual structures are complex, arbitrary, and difficult to visualize
  - Requirements specification must do more than one job well
    - Specification serves many purposes (test-to, design-to, contractual, validation, etc.)
    - Specification has many audiences with different viewpoints, languages, knowledge (customer, designer, tester, regulator, etc.)

Essential Difficulties

- **Control** (decideability, predictability, manageability)
  - *Control* => ability to plan the work, work to the plan
  - Without experience, can't predict which requirements will cause problems
  - Requirements change all the time
  - Together make planning unreliable, cost and schedule unpredictable

Essential Difficulties

- **Inseparable Concerns**
  - Separation of concerns - ability to divide a problem into *distinct* and *relatively independent* parts
  - Many issues in requirements cannot be cleanly separated (I.e., decisions about one necessarily impact another)
    - Need for stability (control) vs. comprehension (need to see it)
    - Formality vs. understanding
  - Implications
    - Difficult to apply "divide and conquer"
      - Cannot achieve effective solutions considering problems in isolation
      - Issue with many "academic" techniques
    - Must make tradeoffs
      - Requires compromises where constraints conflict
      - Differing effects on different stakeholders imply *negotiation*

Accidental Difficulties

- Written as an afterthought: common practice to write requirements after the code is done (if at all)
  - Inevitably a specification of the code as written
  - Not planned, not managed, not reviewed or used
  - Designers and coders end up defining requirements
- Confused in purpose
  - Authors fail to decide precisely what purposes the SRS will serve and in what way it will serve them
  - Requirements end up mixed with other things (overview material, marketing hype, implementation details, etc.)
  - Fails to do anything well
    - Cannot distinguish which parts are really requirements
    - Unclear what tradeoffs to make regarding issues like precision, formality, etc.

Accidental Difficulties

- Not designed to be useful
  - Often little effort is expended on the SRS - and it shows
    - SRS is not expected to be useful - a self-fulfilling prophesy
    - Little effort expended in organizing, writing, checking, managing, or evolving
    - Inherent difficulties doom haphazard approaches to failure
  - Resulting document of limited usefulness
    - Not organized as a technical reference - difficult to look things up
    - No effective procedure for determining consistency or completeness
    - Difficult to make changes, keep consistent
    - Document is difficult to use, quickly out of date
- Lacking essential properties
  - For the SRS to serve its purposes effectively it must have certain properties: Completeness,Consistency, Ease of change, Precision...
  - Accidental difficulties lead to document that is redundant, inconsistent, unreadable, ambiguous, imprecise, inaccurate.
  - Not useful, not used.

Role of a Disciplined Approach

- Meaning of “disciplined”
  - Objectives are known in advance
    - What’s the document’s purpose?
    - Who are in its audience?
  - Product is well defined and designed to purpose
  - Process is defined, followed, tracked and managed
- Disciplined approach will address the accidental difficulties
- Provides a basis for attacking essential difficulties

Address Accidental Difficulties

- Written as an afterthought
  - Plan for and budget for the requirements phase and its products
  - Specification is carefully written, checked, and maintained
- Confused in purpose
  - Document purpose is defined in advance (who will use it and what for)
  - Plan is developed around the objectives
- Not designed to be useful
  - Specification document itself is designed to facilitate key activities
    - document users (e.g., answer specific questions quickly and easily)
    - document producers (e.g., ability to check for properties like consistency)
  - Designed to best satisfy its purpose given schedule and budget constraints
- Lacking essential properties
  - Properties are planned for then built in
  - Properties are verified by the best means possible (review, automated checking, etc.)

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 statue 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 millisec 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 form multiple RT’s, the bus channel's reset capability is inhibited,and the current channel has not bee 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.

CIS 422:
Developing a Good SRS

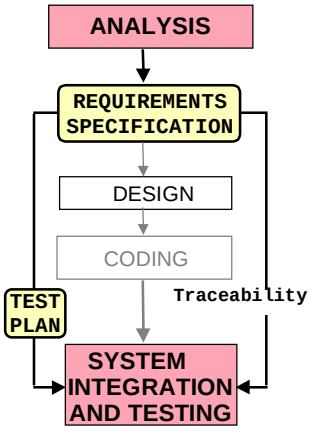
Contents of an SRS

- What should go in a requirements document?
- Everything you need to know to design and write the code
  - No more, no less
  - Every statement should be valid for all acceptable products
  - If a product satisfies every statement, it should be acceptable
  - Areas of incompleteness should be explicit
- Includes non-behavioral attributes like
  - Ease of change or maintenance requirements
  - Safety constraints
  - Reusability constraints

Properties of a Good SRS

- Does an SRS have a “design?” i.e., does it make sense to talk about designing an SRS as a distinct activity from designing the system?
- What would it mean to “design” the SRS?
- What properties should we design for?

Role of Correct Requirements

DEVELOPMENT PHASE	HIGH-ASSURANCE GOAL
	**1. Get the right requirements** **2. Get the requirements right** a. Application-Independent Properties • Consistency • Completeness b. Application-Dependent Properties • Safety properties • Mission-critical properties **3. Verify the system against the requirements** a. Test coverage b. Traceability c. Statistically based testing

Desirable SRS Properties

Semantic Properties*	Packaging Properties+
• Complete • Consistent • Unambiguous • Verifiable • Implementation independent	• Modifiable • Readable • Organized for reference and review • Reusable
* Properties of the specification's semantics (what it says) independently of how it's said	+ Properties arising from the way the specification is written (organization, formats, notations)

Complete

- Formal vs. informal notions of completeness
  - Completeness of a model: everything mentioned is specified
  - Completeness of an SRS: captures all of the customer's requriements
- Complete means:
  - The SRS defines the set of acceptable implementations
  - The SRS contains all the information needed to write the software that is acceptable to the customer (and nothing more)
  - Any implementation that satisfies every statement in the requirements is an acceptable product.
- Why do we need it?
  Why do we need to know when we have it?

Complete (2)

- Why is completeness hard to achieve?
  - Essential:
    - Comprehension (don't know what we want until we see it)
    - Mutability (target keeps moving)
    - Visibility (difficult to measure)
  - Accidental:
    - No objective definitions of completeness
    - Contents of SRS not well defined
    - No guidelines for determining when the specification is done
- Principles
  - SRS contents are determined in advance (i.e., specifics).
  - Methodology provides guidance in answering the question "Am I done yet?"
  - Methodology provides methods for assessing completeness.
  - Where information is not available before development begins, areas of incompleteness must be explicitly indicated.

Consistent

- Consistent means: no two statements of requirements conflict
- Why needed? obvious
- Why difficult to achieve?
  - Accidental difficulty resulting from specifications that are hard to read and review.
  - Often there are poorly documented or understood relationships between requirements.
  - Essential: changes inevitably introduce inconsistencies over time

Unambiguous

- Unambiguous means: there is only one possible interpretation of each requirement.
- Why needed?
  - Everyone agrees on what is to be built and builds what is intended
  - All parts of the development a built with common assumptions
- Why difficult to achieve
  - Essential Dilemma: all natural languages are ambiguous
    - Formal methods are impractical for complex systems
    - Formal methods are essential for precision, unambiguousness
  - Need for practical formal methods
- Principles
  - Be as rigorous as possible in expressing requirements.

Verifiable

- Verifiable means: it is possible to determine unambiguously whether a given implementation satisfies the requirement (also called “testable”).
  - “The system shall be easy to use.”
    vs.
  - “Every menu operation shall be accessible with no more than three mouse clicks.”
- Why needed?
  - Ensures there is an objective measure for each requirement.
  - Allows QA to determine whether or not a given implementation satisfies the requirements (and which ones it satisfies).
- Why difficult to achieve?
  - For system output values, difficulties are typically accidental, arising from poor specification technique.
  - For properties like reliability, useability and fault tolerance, no good mathematical models.

Implementation Independence

- Implementation Independent means:
  - The SRS is free of design and implementation detail
  - … unless those details reflect actual requirements.
- Why needed?
  - Ensures real problem is captured
  - Defines requirements in user's terms
  - Avoids over-constraining design and implementation
- Why difficult to achieve?
  - What vs. how dilemma
  - Developers lack knowledge of problem domain, customers lack knowledge of software engineering
  - Methodology must address these issues.
- Principles
  - Behavioral requirements are written *only* in terms of *externally visible behavior* (i.e., "black-box")

Packaging Properties

Modifiable

- Modifiable means: the specification is relatively easy to change for the set of expected changes
- Why needed? Change disrupts everything.
- Why difficult to achieve?
  - Design cannot be equally easy to change for all changes.
  - Must anticipate likely changes and their importance
- Principles
  - Change is inevitable - plan on it, plan for it!
  - Process and products must support ease of change
    - Analyze problem domain for likely changes (planned improvements and unplanned but expected)
    - Rank changes for likelihood and importance
    - Provide method/ guidance supporting ease of change
    - Provide process for incorporating changes, reviewing results
    - Include likely changes (and related issues) as part of specification
    - Organize document for ease of change (how?)

Readable

- "Readable" means;
  - The intended audience can readily understand the information in the specification
  - The intended audience can readily use the information for its intended purpose

  (e.g., the customer can determine if the system will solve his problem)
- Why needed?
  - An SRS typically has several purposes and audiences who must understand specification
  - Specification that's hard to read won't be properly reviewed or ultimately used
- Why difficult to achieve?
  - Conflicts with need for precision, unambiguousness, testability
  - Difficult to find common languages or points of view.
- Principles
  - Use readable formalisms

Organized (for Reference and Review)

- We assume that the SRS is first and foremost a *technical specification*
  - i.e., it is a reference manual for the expected behavior.
  - It is not: an introduction, an explanation of intended use, a justification of the project
- Organized means: One can use the specification answer specific technical questions quickly and easily.
  - It is clear where each piece of information belongs
  - It is clear where information is incomplete or missing.
- Why difficult?
  - Requires forethought
  - Requires up-front effort, time, and money
  - Essential problem of many audiences and purposes
- Principles
  - One must decide the primary audience and purpose in advance (i.e., what kinds of questions the document will be designed to answer easily).

Reusability

- Reusable means: the effort of developing (analyzing, writing, reviewing, etc.) can be reused rather than redone for a system with overlapping requirements.
- Why needed?
  - Good requirements take a lot of effort, why repeat it unnecessarily?
  - Depends on organization.
- Why difficult?
  - Problem of crystal ball
  - Otherwise, same as mutability.
- Principles
  - Reuse parts common to all projects first (e.g., create common SRS design only once)
  - Apply separation of concerns

Role of Formal Methods

The Formal Methods Dilemma

- Standard approaches (e.g., prose specs) lack sufficient rigor to meet high-assurance goals
- Formal requirements methods have desired technical virtues viewed as impractical for large, complex systems
  - Capability for unambiguous specification, precision, testability, and analyzability
  - Industry concern for practicality
    - Concern for difficulty to write/read, required expertise, ability to scale
    - Concern for real-world issues of fuzzy or changing requirements
    - Concern for fit with industrial development context
    - **Adds up to perceived cost/schedule risk**
- Implication: Need for Practical Formal Methods

What's Missing in Current Approaches?

- Currently have a number of formal **systems** (models)
  - Well defined language, semantics, rules
  - e.g., Z, SCP, VDM
- Formal *System* ≠ Formal *Method*
  - Method should define what to do, when to do it, and why
  - Method should consider cost/benefit of techniques used
  - Most "formal methods" do neither
    - Present formal model, leave context of use to reader
    - Make unrealistic assumptions about when and where formal techniques should be used
    - Make unrealistic assumptions about ease of adoption
- Evidence of effectiveness is largely lacking
  - few real studies of effectiveness (Catch-22)
  - Much available evidence is contra-indicative

The Good News

- 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 tables where possible
- NASA example

Is a “Good” SRS Achievable?

- A qualified “yes”
  - Mutual satisfaction of some goals is difficult
  - Want completeness but users don’t know what they want and requirements change.
  - Many audiences and purposes, only one possible organization and language
  - Want formality (precision, verifiability, analyzability) but need readability.
- Tradeoffs and compromises are inevitable
  - Usefulness of establishing document purpose in advance.
  - Make them by choice not chance!
- It isn’t easy
  - effort
  - expertise
  - technique

    ….but it is necessary!

Documentation Principles

Documentation Principles

- Write specifications for the document
  - Easiest way to ensure agreement on:
    - Intended audience(s)
    - Purpose(s) of the document
    - Document organization or design.
  - **Keep design simple and reusable**
    - Avoid starting another project in specification design
    - …but, do enough to capture consensus and resolve dissention
- Characterize and formulate questions to be answered before starting to answer them
  - Avoids adding extraneous material
  - Avoids answering only the easy questions

Documentation Principles (2)

- Design the document applying *Separation of Concerns*
  - Principle of Separation of Concerns: distinct concerns (i.e., relatively independent issues) belong in distinct (separate) parts of the document
  - Goal is to divide the document into distinct and relatively independent parts
    - Readability: users can read and understand one section of the document without having to read or reference many others.
    - Modifiability: relative independence of the parts means that a changes tend to be confined to one part of the document
    - Reusability: parts that form a distinct whole (i.e. not dependent on other parts) are easier to reuse.

Documentation Principles (3)

- Write once - read many times
  - Remember that the specification will be used many times but only written once (more or less)
  - Best ROI when effort is spent in writing to make it easy to read!
- Formal vs. informal notation
  - Informal (natural language): for introduction, motivation, gaining an overview, or review
  - Formal (mathematics): for binding precise information, ease of reference, ease of checking correctness.

Summary

- Requirements are hard to do well but ...
  - Any specification is better than none
  - Work that goes into requirements is very high leverage (cost ratio)
- Decide in advance
  - Audience: who the document is for
  - Purpose: what the audience will use it for
- Design the document to be useful over the long haul
  - to satisfy the purposes well
  - to be easy to keep up-to-date (easy to modify)
- Make it part of the product development cycle
  - Document is as important as the code (perhaps more so)
  - Plan and budget for it, build to it, maintain it
- Be as formal as possible
  - Use simple formalisms where possible
  - e.g., use prose to explain, tables and expressions to specify detailed behavior