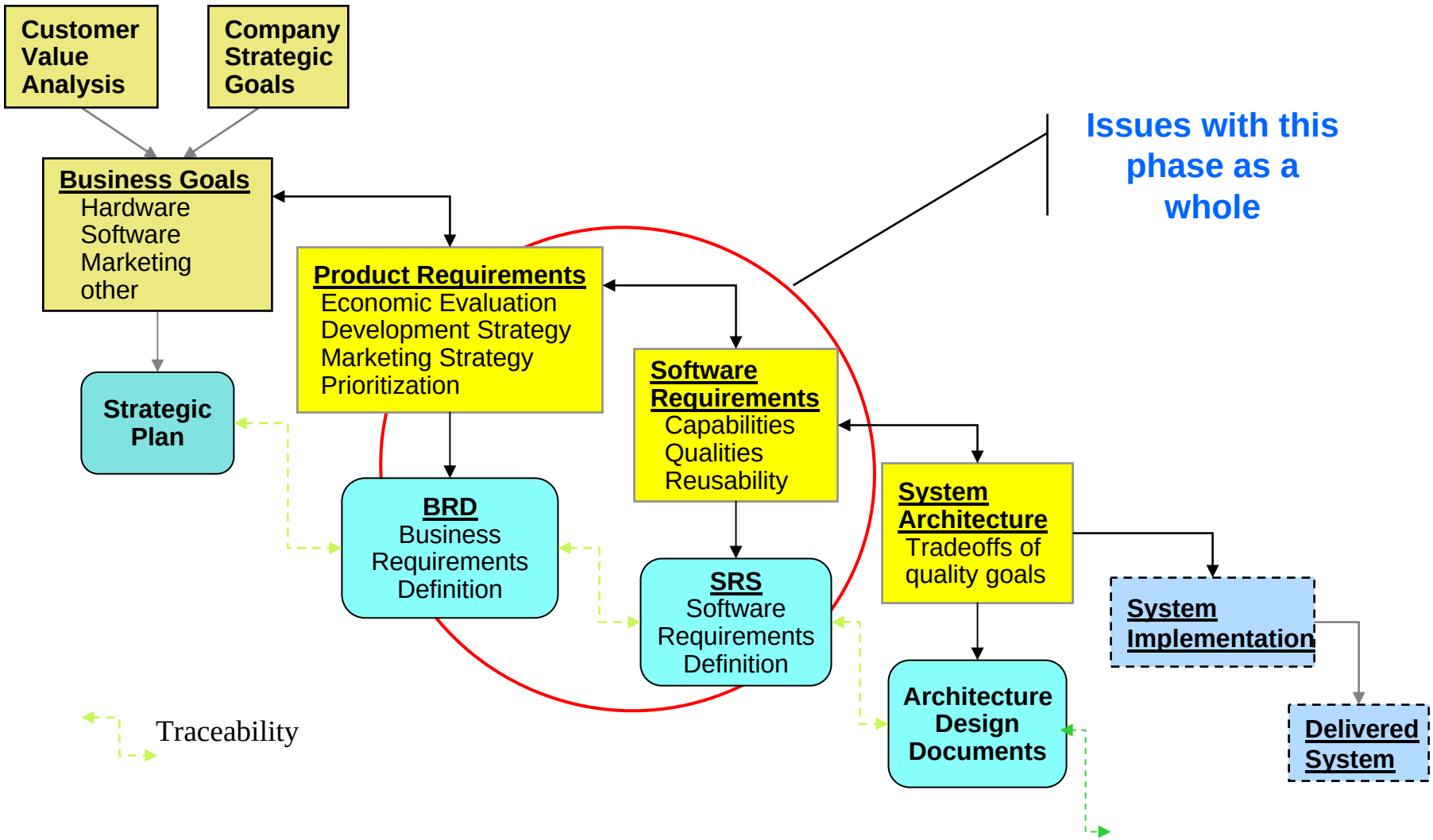
CIS 422/522

Software Requirements I

Overview

- What is a “requirement?”
- 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

Fit in Development Cycle



What is a “software requirement?”

- A description of something the software must do or a property it must have
- Anything the customer says it is!

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.

Many Stakeholders

- For “large” developments, many potential stakeholders
  - 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.

Requirements Phase 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.
- *Specify* in terms of 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.

Two Kinds of Software Requirements

- Concept of Operations: documents user needs and customer expectations
  - Link to organizational goals
  - Stated in terms the the users' / customer's can understand
- Technical Specifications: a concise and unambiguous statement of technical parameters for a system that will satisfy the operational requirements
  - Stated in the developers' terminology
  - Covers issues such as performance, interfaces, safety, security, and reliability
- Issues apply to both but particularly the detailed technical specification

The Trouble with Requirements

“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

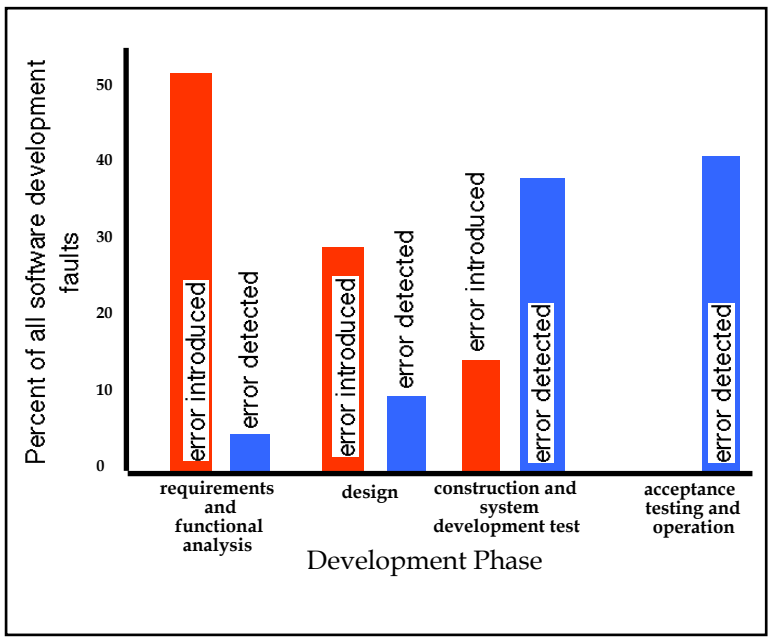
  > Ballistic Missile Defense system developers note: “In nearly every software project that failed to meet performance and cost goals, requirements inadequacies played a major and expensive role in project failure” [Alford 79]

- 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

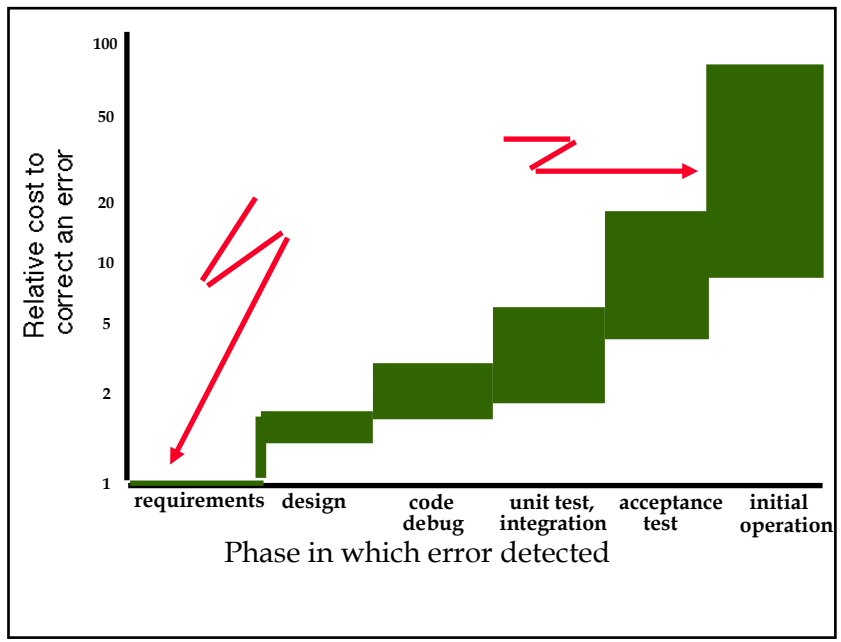
=> SE improvements haven’t kept pace with growth in project scale and complexity

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.



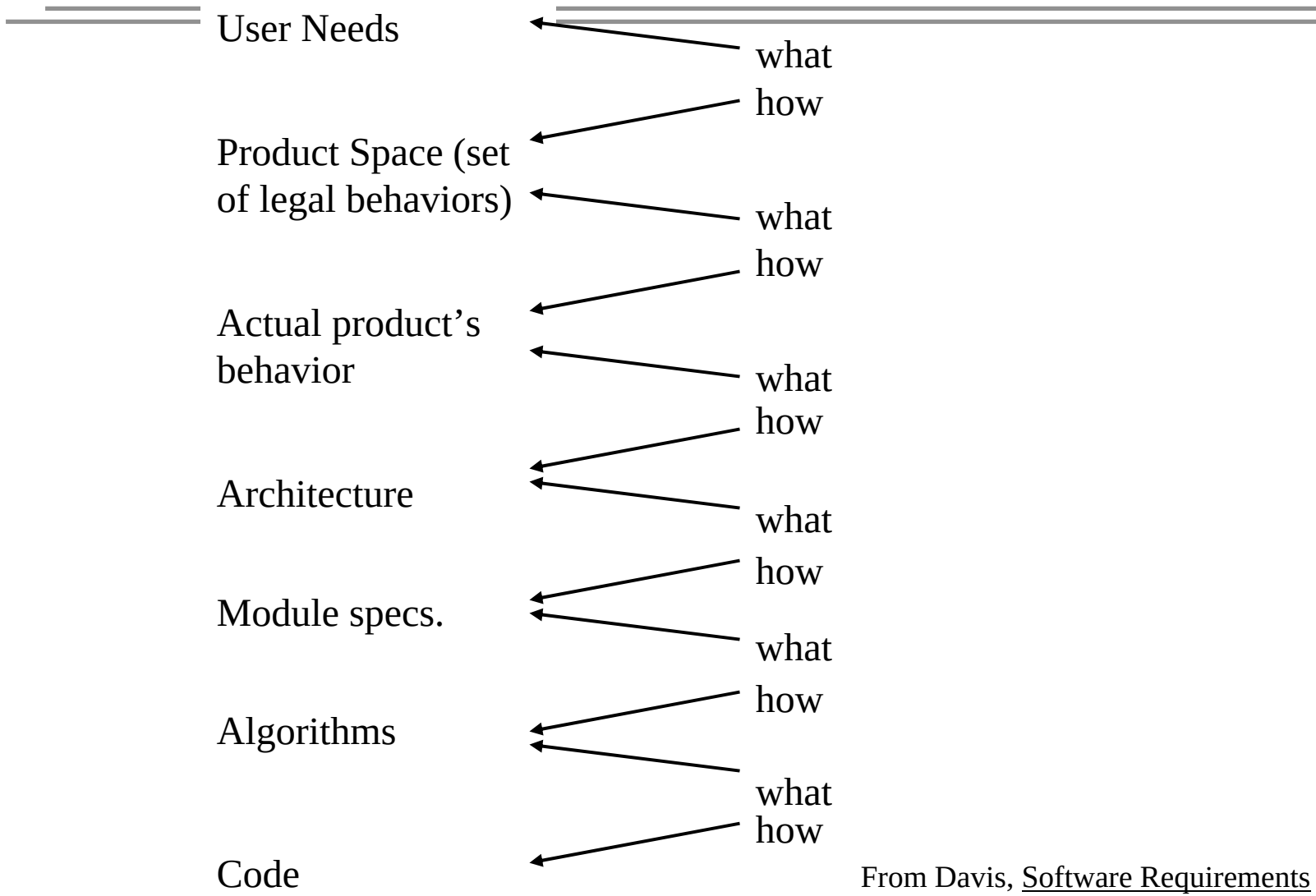
Why do we observe these effects?

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?

Overall Goals of Requirements

- 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

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 (1)

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

Essential Difficulties (2)

- 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 (3)

- 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 (4)

- 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 (1)

- 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

Accidental Difficulties (2)

- 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 (3)

- 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, maintaining
    - 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

Accidental Difficulties (4)

- 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.)

Assignment

- Project presentations on Friday