Midterm Review

The "Software Crisis"

- Have been in "crisis" since the advent of big software (roughly 1965)
- What we want for software development
  - Low risk, predictability
  - Lower costs and proportionate costs
  - Faster turnaround
- What we have:
  - High risk, high failure rate
  - Poor delivered quality
  - Unpredictable schedule, cost, effort
  - Examples: Ariane 5, Therac 25, Mars Lander, DFW Airport, FAA ATC etc.
- Characterized by **lack of *control***

Large System Context

- Discuss issues in terms of large, complex systems
  - Multi-person: many developers, many stakeholders
  - Multi-version: intentional and unintentional evolution
- Quantitatively distinct from small developments
  - Complexity of software rises exponentially with size
  - Complexity of communication rises exponentially
- Qualitatively distinct from small developments
  - Multi-person introduces need for organizational functions, policies, oversight, etc.
  - More stakeholders and more kinds of stakeholders
- We can only approximate this in our projects

Implications: the Large System Difference

- Small system development is driven by technical issues (I.e., programming)
- Large system development is dominated by organizational issues
  - Managing complexity, communication, coordination, etc.
  - Projects fail when these issues are inadequately addressed
- Lesson #1: **programming ≠ software engineering**
  - Techniques that work for small systems fail utterly when scaled up
  - Programming alone won't get you through real developments or even this course

View of SE in this Course

- The <u>purpose of Software Engineering</u> is to *gain* and *maintain* intellectual and managerial control over the products and processes of software development.
  - **Intellectual control** means that we are able make rational choices based on an understanding of the downstream effects of those choices (e.g., on system properties).
  - **Managerial control** means we likewise control development *resources* (budget, schedule, personnel).

Course Approach

- Learn methods for acquiring and maintaining control of software projects
- Managerial control (most of focus to date)
  - Team organization and people management
  - Organizing people and tasks
  - Planning and guiding development
- Intellectual control
  - Choosing appropriate order for decisions and ensuring feedback/correction
  - Establishing and communicating exactly what should be built

Teamwork and Group Dynamics

What do software developers do?

- Most time is not spent coding
- So how do they spend their time?
- IBM study (McCue, 1978):
  - 50% team interactions
  - 30% working alone (coding & related)
  - 20% not directly productive

-Technical excellence is not enough
Must understand how to work in teams

Being a Good Team Member

- Attributes most valued by other team members
  - Dependability
    - When you say you'll do something, you do it
    - Correctly
    - On time
  - Carrying your own weight (doing a fair share of the work)
- People will overlook almost everything else if you do these

Consensus decision making

- Usually best approach for peer teams
- Consensus is not counting votes
  - Democracy is 51% agreement
  - Unanimity is 100% agreement
  - Consensus is neither
    - It is "buying in" by group as a whole, including those who disagree
- Everyone has their say
- Everyone accepts the decision, even if they don't prefer it

Consensus takes time and work, but is worthwhile

The Software Lifecycle

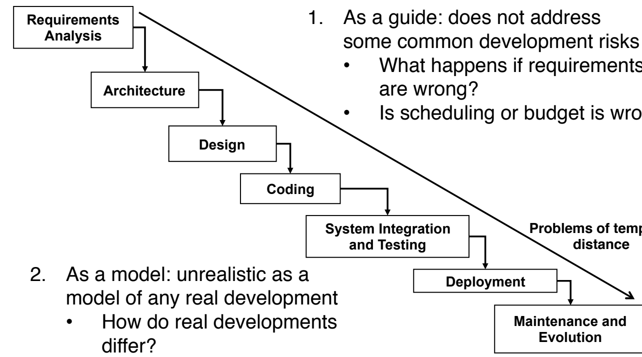
Need to Organize the Work

- Nature of a software project
  - Software development produces a set of interlocking, interdependent work products
    - E.g. Requirements -> Design -> Code
  - Implies dependencies between tasks
  - Implies dependencies between people
- Must organize the work such that:
  - Every task gets done
  - Tasks get done in the right order
  - Tasks are done by the right people
  - The product has the desired qualities
  - The end product is produced on time

Usefulness of Life Cycle Models

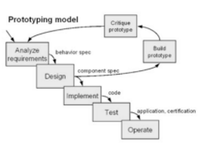
- Application of “divide-and-conquer” to software processes and products
  - Goal: identify distinct and relatively independent phases and products
  - Can then address each somewhat separately
- Intended use
  - Provide guidance to developers in what to produce and when to produce it
  - Provide a basis for planning and assessing development progress
- Never an accurate representation of what really goes on

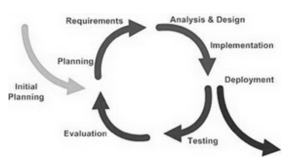
A “Waterfall” Model

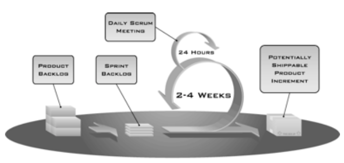


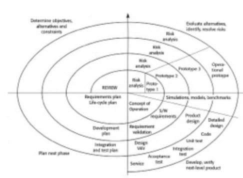
1. As a guide: does not address some common development risks
   - What happens if requirements are wrong?
   - Is scheduling or budget is wrong?
2. As a model: unrealistic as a model of any real development
   - How do real developments differ?
   - Models are abstractions of reality

Other Characteristic Models









Choosing a Process to Use

- Balance goals and risks
- Objective: proceed as systematically as possible from a statement of system goals to an implementation that demonstrably meets those goals
  - Understand that any process description is an idealization
  - Always must compensate for deviation from the ideal (e.g., by iteration)
- Risk: Anything that might lead to a loss of control is a project risk
  - E.g., won’t meet the schedule, will overspend budget, will fail to deliver the proper functionality

A Software Engineering Perspective

- Choose processes, methods, notations, etc. to provide *an appropriate level of control* for the given *product* and *context*
  - Sufficient control to achieve results
  - No more than necessary to contain cost and effort

Example

- Project 1 requirements and constraints
  1. Deadline and resources (time, personnel) are fixed
  2. Delivered functionality and quality can vary (though they affect the grade)
  3. Risks:
     1. Missing the deadline
     2. Technology problems
     3. Inadequate requirements
- Process model
  - All of these risks can be addressed to some extent by building some version of the product, then improving on it as time allows (software & docs.)
  - Technology risk requires building/finding software and trying it (prototyping)
  - Most forms of incremental development will address these

Project Planning and Management

From Process to Plan

- Process definition manifests itself in the project plan
  - Process definition is an abstraction
  - Many possible ways of implementing the same process
- *Project plan makes process concrete*, it assigns
  - People to roles
  - Artifacts to deliverables and milestones
  - Activities to tasks over time
- Project plan should be one of the first products but expect it to evolve

Document Types and Purposes

- Management documents
  - Basis for managerial control of resources
    - Calendar time, skilled man-hours budget
    - Other organizational resources
  - Project plan, WBS, Development schedule
  - Utility: allows managers to track actual against expected use of resources
- Development documents
  - Basis for intellectual control of products (content and qualities)
  - ConOps, Requirements (SRS), Architecture, Detail design, etc.
  - Utility:
    - Vehicles for making and recording development decisions
    - Allows developers to track decisions from stakeholder needs to implementation

Planning Tools

- Review in book
- Work Breakdown Structure: decompose tasks and allocate responsibilities
  - If incomplete, some tasks may not be done
  - If imprecise, people do not know exactly what to do. May do too little or the wrong thing
  - Without a complete set of tasks, schedules are unrealistic
- PERT charts: identify where ordering of tasks may cause problems
  - Represent precedence or resource constraints
  - Identify critical path
- Gantt Charts: method for visualizing project schedule
- Note that these address problems our projects have encountered

Document Types and Purposes

- **Management documents**
  - **Basis for project management (managerial control of resources)**
    - **Calendar time, skilled man-hours budget**
    - **Other organizational resources**
  - **Project plan, WBS, Development schedule**
  - **Use: allows managers to track actual against expected consumption of resources**
- Development documents
  - Basis for intellectual control
    - Used for making and communicating engineering decisions (requirements, design, implementation, verification, etc.)
    - Allows developers to track decisions from stakeholder needs to implementation
  - Basis for communicating decisions
  - ConOps, SRS, Architecture, Detail design, etc.

Requirements

Problem Analysis
Requirements Specification

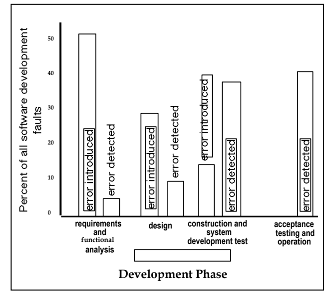
What is a "software requirement?"

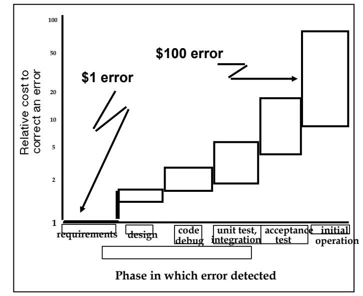
- A description of something the software must do or property it must have
- The set of system requirements denote the problem to be solved and any constraints on the solution
  - Ideally, requirements specify precisely what the software must do without describing how to do it
  - Any system that meets 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





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, observed this and the resulting problems in projects
- Understanding what makes these goals difficult to accomplish helps us understand how to mitigate the risks

What makes requirements difficult?

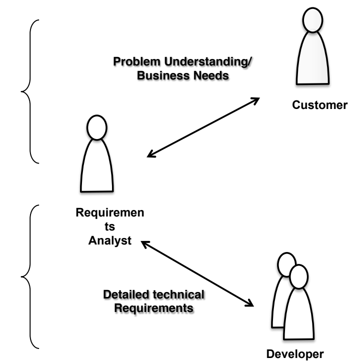
- Comprehension (understanding)
  - People don't (really) know what they want (…until they see it)
  - Superficial grasp is insufficient to build correct software
- Communication
  - People work best with regular structures, conceptual coherence, and visualization
  - Software's conceptual structures are complex, arbitrary, and difficult to visualize
- Control (predictability, manageability)
  - Difficult to predict which requirements will be hard to meet
  - 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)
  - Difficult to apply "divide and conquer," must make tradeoffs
- Implies all the requirements goals are difficult to achieve

Purposes and Stakeholders

- Many potential stakeholders using requirements for different purposes
  – Customers: the requirements document what should be delivered
  – 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
  – Quality Assurance: basis for validation, test planning, and verification
  – Also: potentially Marketing, regulatory agencies, etc.

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.



Documentation Approaches

- ConOps: informal requirements to describe the system’s capabilities from the customer/user point of view
  – 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
- SRS: formal, technical requirements for development team
  – 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”
  – Precise, unambiguous, complete, and consistent as practical

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.
- Exemplified by discussion of use cases

Scenario Analysis and Use Cases

- Applying scenario analysis in the development 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)

Use Cases

1 Use Case: Manage Reports

1.1 Description

This Use Case describes operation 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

etc

A systematic approach to use cases

- **Uses a standard template**
- **Easier to check, read**
- **Still informal**

Benefits and Drawbacks

- Use cases can be an effective tool for:
  - Identifying key users and their tasks
  - Characterizing how the system should work from each user’s point of view
  - Communicating to non-technical stakeholders
- Generally inadequate for detailed technical requirements
  - Difficult to find specific requirements
  - Inherently ambiguous and imprecise
  - Cannot establish completeness or consistency
  - Possible exception: applications doing simple user-centric tasks with little computation (e.g., your project)

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
  - Can systematically check completeness consistency
- Provides
  - Detailed specification of precisely what to build
  - Design-to specification
  - Build-to specification for coders
  - Characterizes expected outputs for testers

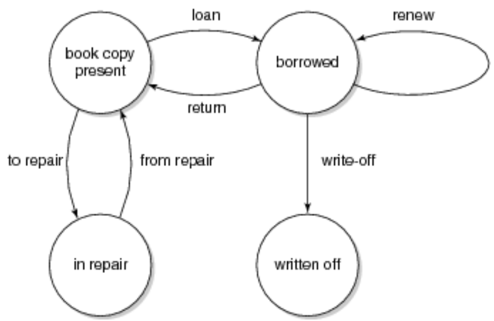
Desirable SRS Properties

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)

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

Graphic Notations for Simple Machines



Real meaning of "control"

- What does "control" really mean?
- Can we really get everything under control then run on autopilot?
- Rather control requires continuous feedback loop
  1. Define ideal
  2. Make a step
  3. Measure deviation from idea
  4. Correct direction or redefine ideal and go back to 2

Questions?