CIS 422/522

Project Planning

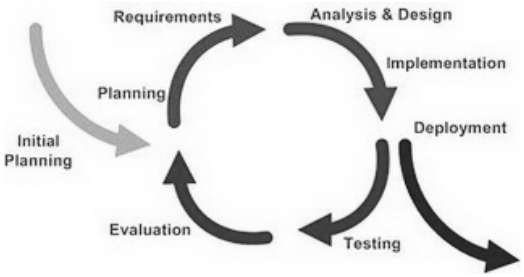
1

Review: 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 -> Test
  - 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

Use Iterative Process Model

- Process viewed as a sequence of iterations
- Addresses key risks
  - Have something to deliver
  - Feedback loop built in
- Each team will implement the abstract model differently



From Process to Plan

- Process 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* is itself a product of the process
  - Activity: project planning
  - Artifact: the Project Plan
  - Roles: Project Manager (owner), team members
- Evolves as the project proceeds

Project Plan

- Purpose: specifies how project resources will be organized to:
  - Create each deliverable
  - Meet quality goals
  - Address developmental goals (e.g., mitigate risk)
- Audience: answers specific kinds of questions for specific types of users, e.g.:
  - General stakeholders: What is the development approach? How does it address project risks?
  - Customers: When will the product be delivered?
  - Managers: When will tasks be completed? What is the current progress against the plan?
  - Developers: What should I be working on and when?

Plan Outline

- Plan contents (template in Assembla workspace)
  - Purpose and audience (who will use the document?)
  - Project background (from requirements)
  - Team roles and responsibilities
  - Risks and risk mitigation
    - What are the key risks? (Team should brainstorm this)
    - Which mitigation strategies will the project deploy
  - Process: development process, how its tailored, rationale
  - Mechanisms, methods, and techniques
    - What kinds of methods and tools will be used?
    - E.g., planning tools, design methods, IDEs, etc.
  - Detailed schedule and milestones
  - Resources and references

Your Project Plans

- ***This is not an abstract, hypothetical exercise!***
- Your projects have real
  - Resources (people, time)
  - Risks (schedule, quality, etc.)
  - Process, schedule, etc.
- These must be reflected in your meetings, plans, schedules, and other work products
- This is how you demonstrate mastery of class concepts

Detailed Schedule and Milestones

- Maps people to tasks over time such that
  - Personnel are fully engaged (time is not wasted)
  - Delivery meets schedule
- Answers: “Who is working on which tasks, what is their progress, and when will they be finished?”
- Inputs
  - Set of artifacts to be created (superset of deliverables)
  - Dependencies/precedence between tasks
  - People filling roles that perform tasks
  - Time budget for each task
- Output
  - Current project schedule
  - Deadline for each task
  - Sequencing among tasks
  - Allocation of people to tasks

Project Plan Template

- Use the template provided in your Assembla team workspace
- This should be a *living document*
  - Changed as the project progresses
  - Ideally, always gives a current view of the *progress against the plan*
    - Shows planned activities
    - Gives snapshot of the current project state
    - This is what I am looking for (or any manager)

Project Planning Tools

Work Breakdown Structure (WBS)

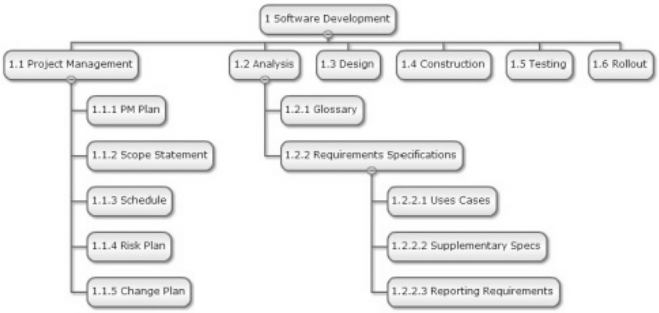
PERT Chart

Gantt Chart

Work Breakdown Structure

- Structured technique for decomposing work into individual tasks with the goals:
  - Identify the complete set of tasks in the project
  - Provide units of work (for individuals or teams)
  - Provide units of work for scheduling and costing
- Identify hierarchy of tasks and subtasks
  - Identify major tasks in project
  - Decomposing each element into component parts
  - Continuing to decompose until manageable work packages can be mapped to roles
- Works best when:
  - Tasks correspond to key deliverables
  - Sum of tasks is 100% of the work
  - Tasks do not overlap
  - Each leaf task takes about the same amount of time

Work Breakdown Structure



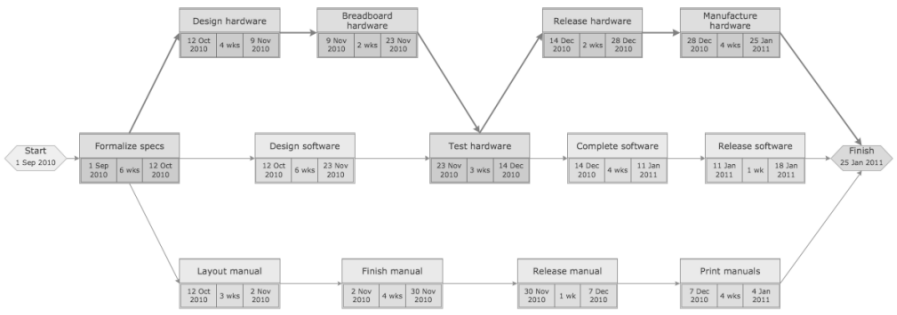
1. Software Development
   1. Project Management
   2. Analysis
      1. Glossary
      2. Requirements Specification
         1. Use Cases
         2. Supplementary Specs…

Equivalent list format

Pert Chart

- Network analysis or PERT is used to identify dependencies between the tasks in the work breakdown structure
- Helps identify where ordering of tasks may cause problems because of precedence or resource constraints
  - Where task B cannot begin before task A ends
  - Where one person cannot do two tasks at the same time
  - Where adding a person can allow tasks to be done in parallel, shortening the project

PERT Chart Example



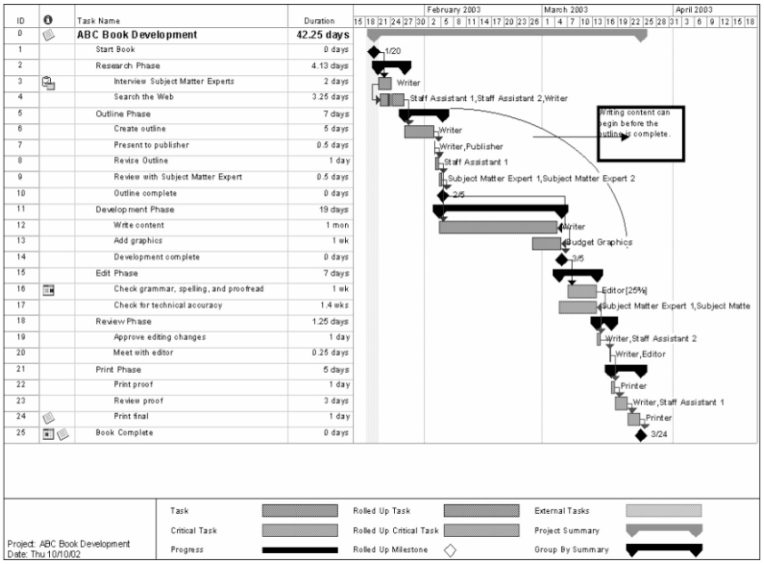
- Which tasks can we start on?
- Which tasks can be done concurrently?
- Which tasks depend on which other tasks?
- *Critical Path*: dependency path with the longest duration
  - Gives minimum length of project

http://www.conceptdraw.com/samples/project-chart

Gantt Charts

- Method for *visualizing a project schedule* in one chart showing
  - The set of tasks
  - Start and completion times
  - Task dependencies
  - Responsibilities
- PERT charts can be reformatted as Gantt charts
- Typically requires a tool, e.g., http://www.ganttproject.biz/, smartchart

Example Gantt Chart



Project Milestone Planning

- Milestone planning is used to show the major steps that are needed to reach the goal on time
- Milestones typically mark completion of key deliverables or establishment of baselines
  - *Baseline*: when a work product is put under configuration management and all changes are controlled
- Often associated with management review points
  - E.g., Requirements baseline, project plan complete, code ready to test
- Can use Gantt or PERT charts to show milestones
- Begin with project events in Schedule

A Simple Alternative

Week 1:

Date Assigned	Due Date	Task	Person Responsible	Status	Date Completed
2/3	2/5	Brainstorm project ideas	Everyone	Complete	2/5
2/3	2/4	Set up meeting w/ instructor	Heidi	Complete	2/3
2/3	2/6	Decide on project	Everyone	Complete	2/6
2/6	2/10	Create Git repository	Heidi	Complete	2/6

Week 2:

Date Assigned	Due Date	Task	Person Responsible	Status	Date Completed
2/10	2/10	Decide on software requirements	Everyone	Complete	2/10
2/10	2/15	Plan and design 1st iteration	Everyone	Complete	2/13
2/10	2/10	Set up meeting w/ Kathleen Freeman-Hennessy	Heidi	Complete	2/10
2/13	2/15	Write ConOps	Nicole, Heidi	Complete	3/2
2/13	2/19	Write project plan	Nicole, Heidi	Complete	2/19
2/13	2/22	Write software requirements	Nicole, Heidi	Completed	3/2
2/15	2/24	Implement 1st iteration	Dex, Hans, Yakun	Complete	2/24

How much planning?

- Planning itself consumes resources; how much planning is enough?
- Enough that:
  - Everyone knows what they should be doing
  - Everyone knows what other people are supposed to be doing
  - Everyone knows when specific deliverables should be finished
    - Can track dependencies between their tasks and others
    - Know when task inputs will be available
  - It is easy to determine the current status of the project against plan
- In practice, detail decreases with distance

Summary

- Project plan makes process concrete
  - People to roles
  - Artifacts to deliverables and milestones
  - Activities to tasks over time
- Plan is key to organizing the work but expect it to change
  - *The plan is nothing, the planning is everything* – D. Eisenhower
- Should understand the use of common planning tools (WBS, Pert, Gantt)

Questions?

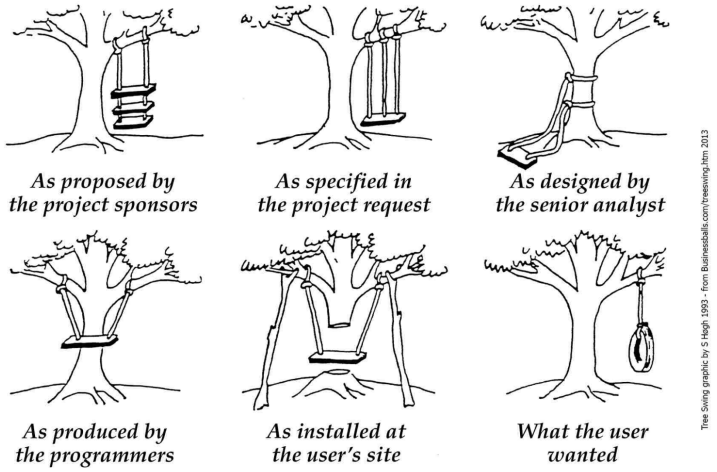
CIS 422/522

Software Requirements 1

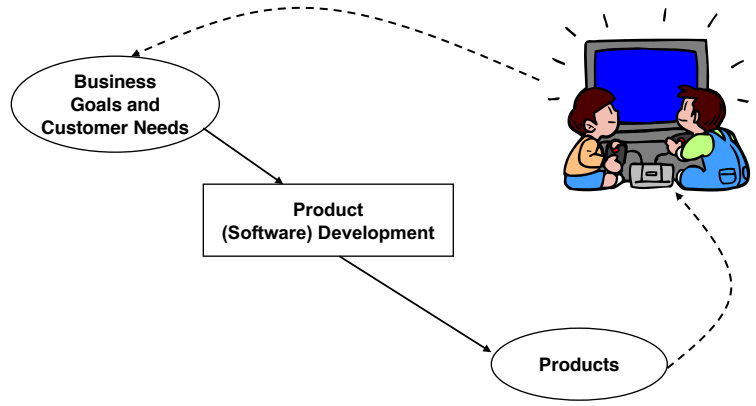
Stuart Faulk
Computer and Information Science

Understanding Software Requirements (and why we get it wrong so often)

"Problem solving is an art form not fully appreciated by some"

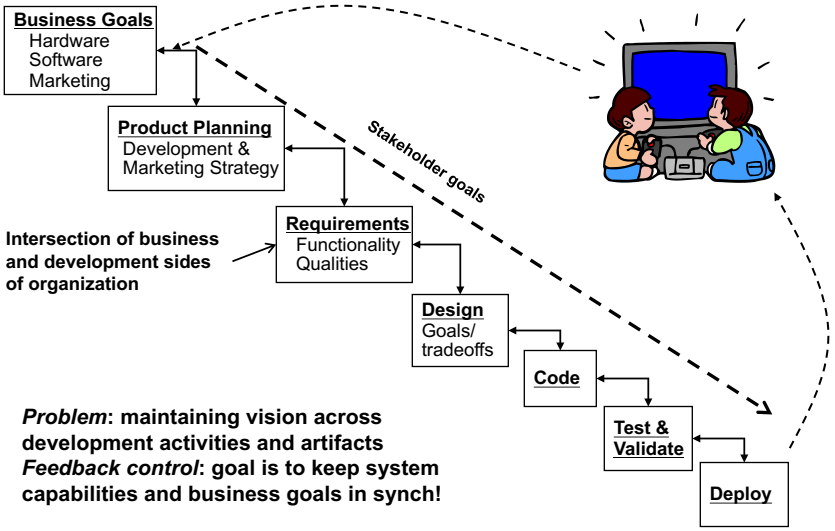


10,000 ft. View



What should the development process accomplish?

Product Development Cycle



***Problem*: maintaining vision across development activities and artifacts**
***Feedback control*: goal is to keep system capabilities and business goals in synch!**

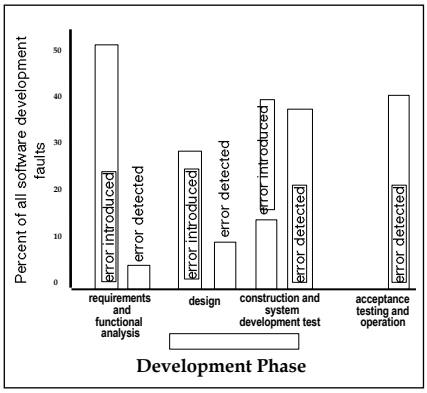
What is a “software requirement?”

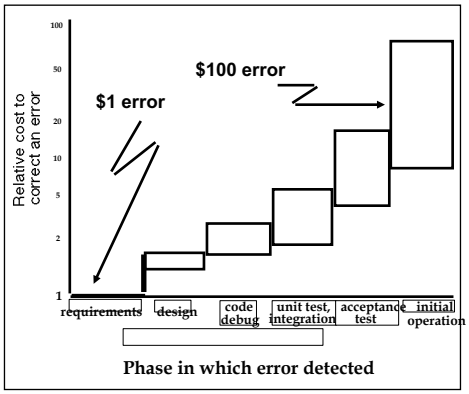
- *Definition*: A description of something the software must do or property it must have
- The set of system requirements denote the problem to be solved and any constraints on the solution
  - Ideally, requirements specify precisely what the software must do without describing how to do it
  - Any system 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
- Understanding what makes these goals difficult to accomplish helps us understand how to mitigate the 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
  - 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 where requirements conflict

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
- All three goals are inherently difficult
- Must be managed as risks

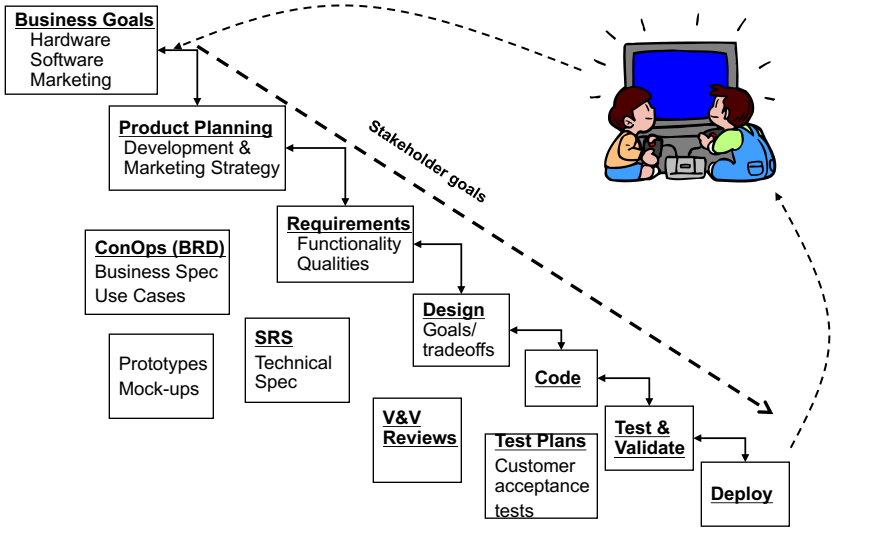
Requirements Process

Understand, Communicate & Control

A good process helps manage requirements difficulties requires having

1. Requirements Understanding (Understand)
   - Elicitation - How do we establish "what people want?"
   - Negotiation - How do we resolve stakeholder conflicts?
2. Requirements Specification (Communicate)
   - Concept of Operations (ConOps) - How do we communicate with non-programmer audiences?
   - Software Requirements Specification (SRS)- How do we specify precisely what the software must do?
3. Requirements V&V (Control)
   - Validation- How do we establish that we have the right requirements?
   - Verification - How do we establish that the implementation is consistent with the specification?

Related Products



1.1 Elicitation

- Goal: Understand precisely what is required of the software
  - Answer the question, “What do the stakeholders want?”
  - Stakeholder: anyone with a valid interest in the outcome of a software development
- Inherently open-ended, ambiguous question
- Addressed by a number of elicitation methods
  - Interview – traditional standard
  - Focus groups
  - Prototyping
  - Use cases
- All have differing costs, strengths, and weaknesses. None is a complete solution
  - Use more than one approach
  - Check the results *early and often*

1.2 Requirements Negotiation

or “Why the customer is not always right!”

•Stakeholders’ requirements often conflict

- Needs of different customers/users may conflict
  - E.g., Salesmen want convenience and speed, management wants security and accountability
- Developer’s needs may conflict with customer’s
  - E.g., development cost vs. customer desires

•Choosing which requirements should be addressed and their relative importance requires *negotiation* and *tradeoffs* among stakeholders

2. Requirements Specification

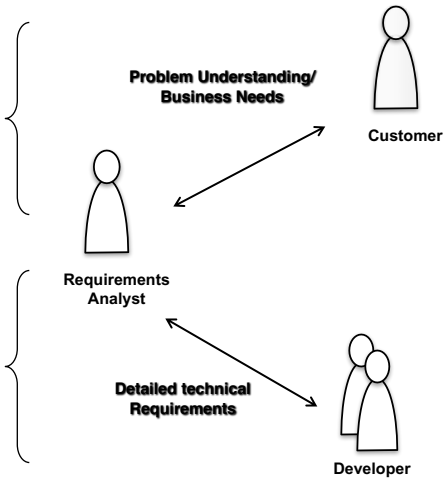
- Goal: Communicate requirements understanding to all system stakeholders
- Q: What kinds of information need to be communicated?
  - System context (link to business objectives)
    - System stakeholders
    - Product business goals
    - System purpose
    - Interfacing systems (if any)
  - System detailed requirements
    - Behavioral requirements
    - Quality requirements

SRS Purposes and Stakeholders

- Sits at business/development intersection
- Many potential stakeholders using requirements for different purposes
  - Customers: document what should be delivered
  - Marketing: capabilities to 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 and is the final authority on the outputs that must be produced
  - Quality Assurance: basis for validation, test planning, and verification

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 Requirements Documentation

- Communicate with stakeholders who understand the problem domain but not necessarily programming:
  - e.g. customers, users, marketing
  - Must develop understanding in common language
  - Role of ConOps (Concept of Operations)
- Communicate with developers
  - Stated in the developer's terminology
  - Sufficiently precise and detailed to code-to, test-to, etc.
  - Addresses properties like completeness, consistency, precision, lack of ambiguity
  - Role of SRS (Software Requirements Specification)
- For businesses, these may be two separate documents

SRS Template

1. Introduction

1.1 Intended Audience and Purpose

<Describes the set of stakeholders and what each stakeholder is expected to use the document for. If some stakeholders are more important than others, describes the priorities.>

1.2 How to use the document

<Describes the document organization. This section should answer for the reader: "Where do I find particular information about X?">

Informal, user centric

2. Concept of Operations

<Use this section to give a detailed description of the system requirements from a user's point of view. The ConOps should be readable by any audience familiar with the application domain but not necessarily with software. The ConOps should make clear the context of the software and the capabilities the system will provide the user.>

2.1 System Context

<Specify the system boundaries including, particularly, the inputs and outputs. May include an illustration or context diagram. >

2.2 System capabilities

<System capabilities may be described in prose or with informal scenarios.>

3. Behavioral Requirements

<Specification of the observable system behavior.>

3.1 System Inputs and Outputs

3.2 Detailed Output Behavior

<A black box specification of the visible, required behavior of the system outputs as a function of the system inputs. Tables, functions, use cases or other methods of specification may be used.>

Formal, technical

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?"
  - Focus on communication over rigor
- More formal, technical requirements for development team (architect, coders, testers, etc.)
  - Purpose is to answer specific technical questions about the requirements quickly and precisely
    - "What should the system output for this set of inputs?"
    - Reference, not a narrative, does not "tell a story"
  - Goal is to develop requirements that are precise, unambiguous, complete, and consistent
  - Focus on precision and rigor (may use formal languages)
  - We will only do a little of this

Informal Specification 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.

Example: Use Cases

- Informal specification requirements in terms of system capabilities provided to a user
- Each Use Case describes how the system and a user interact to accomplish a user task
  - Specifies (only) functional behavior
  - Captures the “business logic” of the application
- Inherently ambiguous, incomplete
  - Can add rigor with consistent templates, good process, reviews

1 Brief Description

This use case describes how the Bank Customer uses the ATM to withdraw money to his/her bank account.

2 Actors

2.1 Bank Customer
2.2 Bank

3 Preconditions

There is an active network connection to the Bank.
The ATM has cash available.

4 Basic Flow of Events

1. The use case begins when Bank Customer inserts their Bank Card.
2. Use Case: Validate User is performed.
3. The ATM displays the different alternatives that are available on this unit. [See Supporting Requirement SR-xxx for list of alternatives]. In this case the Bank Customer always selects "Withdraw Cash".
4. The ATM prompts for an account. See Supporting Requirement SR-yyy for account types that shall be supported.
5. The Bank Customer selects an account.
6. The ATM prompts for an amount.
7. The Bank Customer enters an amount.
8. Card ID, PIN, amount and account is sent to Bank as a transaction. The Bank Consortium replies with a go/no go reply telling if the transaction is ok.
9. Then money is dispensed.
10. The Bank Card is returned.
11. The receipt is printed.

5 Alternative Flows

5.2 Wrong account

If in step 8 of the basic flow the account selected by the Bank Customer is not associated with this bank card, then

1. The ATM shall display the message "Invalid Account – please try again".

2. The use case resumes at step 4.|

Example Use Case

- Avoids design decisions
- References other use cases
- References more precise definitions where necessary
- Some terms need further definition (e.g. PIN)

Supports Requirements Goals

Applying Use Cases in the requirements process

- Requirements Elicitation
  - Identify which capabilities the system should provide to each class of users
  - Collect in terms of problem domain goals
  - Provides basis for prototypes, mockups
- Requirements Communication (ConOps)
  - Record as use-cases with standard format
  - Easy to read and understand for wide audience
- Requirements verification and validation
  - Review use-cases for consistency, completeness, user acceptance
  - Create test cases consistent with use cases
  - Verify against code (e.g., use-case based testing)

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
  - Elicitation and validation activities
  - Specification activities
  - Verification and validation activities
  - Requirements change management

End

CIS 422/522

Use Case Summary
In-class Exercise

Project Planning Notes

- Work assignments
  - Tendency to be vague about what must be done
    - Tasks open to interpretation
    - Results often not what is wanted
  - Should tie to specific deliverables, quality goals
    - Use grading rubric, examples
- Use scheduling to make sure:
  - Every task is owned and tracked
  - Key milestones accounted for
  - Every team member is adding to progress
- Together make sure all tasks are accounted for (especially non-coding tasks)

Exercise: Plan to Bake a Cake

- How many cooks does it take to bake a cake?
  - Can more people produce a cake faster?
  - Is there a limit to how fast?
  - If you want the cake ready for a party at 4:00 PM, how late can you start?

USE CASES

Problems

- How to convey typical usage scenarios to stakeholders in a way that all can understand
  - Customers, marketers, architects, developers, testers
  - Provide a lightweight means for exploring requirements
- How to quickly express key requirements for users in a standardized way
- How to provide a basis for system testing
- How to identify issues for prototyping
- How to start thinking about traceability from requirements to architecture

"Use Cases" can be an effective technique

Use Cases

- Use Case: a narrative describing how the system and a user interact to accomplish a user task
- A form of User Centered Analysis – capturing requirements from the user's point of view
  - Identify capabilities required by different types of users (customer, administrator, etc.)
  - Includes only user-visible functional requirements

Identifying Actors

- Actors – identifies the roles different users play with respect to the system
  - Roles represent classes of users with different goals
  - Actors carry out use cases
- Helps identify requirements for different kinds of users
  - "How would depositors use the system?"
  - "How would a library patron use the system?"
- Diverse classes of users may require different interfaces
  - E.g., users vs. administrators vs. content providers

Scenario Elicitation

- Each class of actor is interviewed and/or observed
  - How do you do task T?
  - How will the user interact with the system to do X?
- Collect in the form of use cases
  - Document in loose text or standard format
  - Identify relative priorities of tasks
  - Resolve conflicts, tradeoffs

Creating Use Cases (Basic)

- Identify a key *actor* and *purpose*
  - The purpose informs the use case title and description
- Identify the main flow (ideal path) from the starting point to the result
  - Preconditions: anything that must be true to initiate the Use Case
  - Trigger: event, if any, initiating the Use Case
  - Basic Flow: sequence of interactions from the trigger event to the result
  - Alternative Flows: identify sequences branching off the Basic Flow
  - Exceptions: identify responses to error conditions

1 Brief Description

This use case describes how the Bank Customer uses the ATM to withdraw money to his/her bank account.

2 Actors

2.1 Bank Customer
2.2 Bank

3 Preconditions

There is an active network connection to the Bank.
The ATM has cash available.

4 Basic Flow of Events

1. The use case begins when Bank Customer inserts their Bank Card.
2. Use Case: Validate User is performed.
3. The ATM displays the different alternatives that are available on this unit. [See Supporting Requirement SR-xxx for list of alternatives]. In this case the Bank Customer always selects "Withdraw Cash".
4. The ATM prompts for an account. See Supporting Requirement SR-yyy for account types that shall be supported.
5. The Bank Customer selects an account.
6. The ATM prompts for an amount.
7. The Bank Customer enters an amount.
8. Card ID, PIN, amount and account is sent to Bank as a transaction. The Bank Consortium replies with a go/no go reply telling if the transaction is ok.
9. Then money is dispensed.
10. The Bank Card is returned.
11. The receipt is printed.

5 Alternative Flows

5.2 Wrong account

If in step 8 of the basic flow the account selected by the Bank Customer is not associated with this bank card, then

1. The ATM shall display the message "Invalid Account – please try again".

2. The use case resumes at step 4.|

Example Use Case

- Avoids design decisions
- References other use cases
- References more precise definitions where necessary
- Some terms need further definition (e.g. PIN)

Guidelines for Good Use Cases

- Use Cases should express requirements, not design or implementation
  - Focus on important *results* that provide *value* to specific actors
    - I.e., if nobody really cares about the outcome, it is not a good use case
  - Focus on *what* the actor is doing, not the details of *how*
    - Not: "The user left-clicks on the radio button labeled *Balance* and presses the *Enter* button"
    - "The user elects the option to view the balance."
- Looking for a small number of use cases that capture the most important interactions
  - Read the IBM Use Case paper

Scenario Analysis Process

Applying scenario analysis in the requirements process

- Requirements Elicitation
  - Identify stakeholders who interact with the system (actors)
  - Collect "user stories" - how people would interact with the system to perform specific tasks
- Requirements Communication (ConOps)
  - 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
  - Combine with mock-ups or prototypes
  - Verify against code (e.g., use-case based testing)

Questions?

Deliverables Walkthrough

- Consider: What kinds of questions should your documents answer?
  - Assume a manager unfamiliar with the project is reviewing your status
  - Would your documents answer key questions about the project goals and current status?
- *Team page*: Who is on the team and what are their skills?
- *Project plan*
  - Who is responsible for which tasks?
  - What are the anticipated risks and what are you doing to mitigate them?
  - What is your development process and how does it help address the risks?
  - Detailed Schedule & Milestones
    - What is the project schedule of tasks and deliverables?
    - What is the current status relative to schedule?

Walkthrough (2)

- *Software Requirements*
  - 2. ConOps: What capabilities will the software provide the user or customer?
  - 3. Behavioral Requirements: What are the detailed technical requirements?
    - Specific inputs accepted & outputs generated
    - Detailed behavior of any computation (e.g., sort, error responses)
  - 4. Quality Requirements: objective requirements for software qualities (e.g., reliability, performance)
- *Software Design*
  - Architecture: How is the software organized into components? How does it work (function)? Where is each requirement implemented (traceability)?
  - Module Interfaces: What are the component interfaces?

Walkthrough (3)

- *Quality Assurance*: How will you check whether the software satisfies functional and quality requirements?
  - Reviews: Which artifacts/properties will be checked by review?
  - Test Plans: How will you test the software?
- *User Documentation*: How will users understand how to install and use the application?
- *Code Documentation:* What do I need to know to find parts of the code responsible for implementing any given requirement or part of the design?
  - How is the code organized in the repository?
  - What does this code component do?

CIS 422/522

Technical Requirements (SRS)
Quality Requirements

1

Review: Use Cases

- Natural language narrative describing how a user interacts with the system to accomplish a specific task
- Defines a *subset of functional requirements*
  - Captures requirements visible to the user
  - Focuses on most important user tasks
- Missing specifications of quality requirements (performance, security, etc.)

Requirements Documentation

- Is a more formal, detailed requirements specification necessary?
- How do we know what "correct" means?
  - How do we decide exactly what capabilities code 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

- Focuses on developing a rigorous specification
  - Should be straight-forward to determine acceptable inputs and outputs
  - Preferably, can systematically check completeness consistency
- Use cases are not sufficient
- Generally accomplished by *modeling* required behavior
  - Formal model: models based on formal languages
  - Partial and semi-formal models

Formal Models

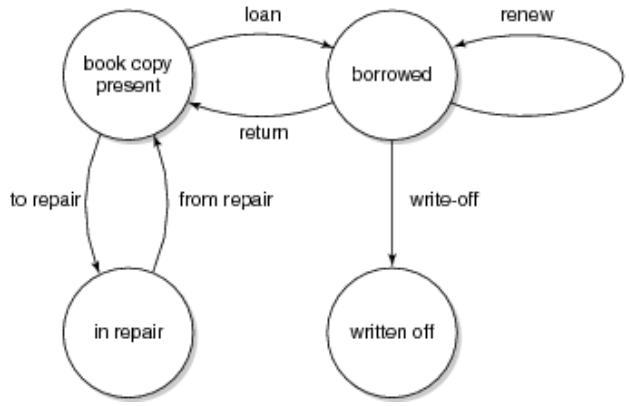
- Requirements modeling methods based on formal languages, e.g.
  - SCR: finite state machines
  - Z: formal logic
  - Statecharts: concurrent automata
- Advantages: allows users to
  - Derive the set of acceptable outputs for given inputs
  - Prove properties like consistency, completeness, safety, liveness
- Disadvantages
  - Requires rare skills
  - Expensive to produce and change
- Used seldom except where mission/safety critical (e.g., Intel fab after $475M FDIV error)

Semi-formal Modeling

- Many semi-formal methods used
  - Structured but non-mathematical models
  - Formal but partial models
- E.g. UML models add some rigor to Use Cases
  - Activity diagrams
  - Sequence diagrams
  - Disadvantage: tends to model design and implementation
- Modeling critical parts of the requirements
  - Use predicates (i.e., basic Boolean expressions)
  - Use mathematical expressions
  - Use tables
- 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
  - Often easier, less time consuming than trying to say the same thing in prose

Example state transition diagram

SE, Modeling, Hans van Vliet, ©2008



Does the Address Book have stateful behavior?
What are the states? Transitions?

Formal Specification Example

Type Dictionary

Name	Base Type	Units	Legal Values	Comment
Speed	Integer	Knots	[0, 250]	Speed measured in nautical miles per hour.
Weight	Integer	percent	[0,100]	Weighting for weighted average
time	Integer	seconds	time > 0	Time in seconds.

Monitored Variable Dictionary

Name	Type	Initial Value	Accuracy	Comment
LowResWS1	Speed	0	1	Wind speed reported by first low resolution sensor
LowResWS2	Speed	0	1	Wind speed reported by second low resolution sensor
HighResWS1	Speed	0	2.5	Wind speed reported by first high resolution sensor
HighResWS2	Speed	0	2.5	Wind speed reported by second high resolution sensor

Controlled Variable Dictionary

Name	Type	Initial Value	Accuracy	Comment
TransmWindSpeed	MsgType	ShortMsg	N/A	Transmitted value of wind speed

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

For Your Projects

- Inputs and outputs
  - Be explicit about value types and ranges for each input variable (e.g. Name, Zip, phone)
    - How many digits? Other characters?
  - Be explicit about acceptable outputs
    - Export values and formats
    - Values displayed or printed
  - Easiest to define the inputs and outputs as abstract variables
- Detailed behavioral requirements
  - Specify acceptable results for a sort
  - Specify acceptable search results
  - Specify state changes (if applicable)

Quality Requirements

Terminology

- Avoid “functional” and non-functional" classification
- Behavioral Requirements – any information necessary to determine if the run-time behavior of a given implementation constitutes an acceptable system
  - All quantitative constraints on the system's run-time behavior
  - Other objective measures (safety, performance, fault-tolerance)
  - In theory all can be validated by observing the running system and measuring the results
- Developmental Quality Requirements- any constraints on the system's static construction
  - Maintainability, reusability, ease of change (mutability)
  - Measures of these qualities are necessarily relative (I.e., in comparison to something else)

Behavioral vs. Developmental

Behavioral (observable)	**Developmental Qualities**
• Performance • Security • Availability • Reliability • Usability	• Modifiability(ease of change) • Portability • Reusability • Ease of integration • Understandability • Support concurrent development
Properties resulting from the behavior of components, connectors and interfaces that exist at run time.	Properties resulting from the structure of components, connectors and interfaces that exist at design time *whether or not they have any distinct run-time manifestation*.

Specifying Quality Requirements

- Is it important to specify the quality requirements explicitly? Unambiguously?
  - Hint: what role would quality requirements play in customer acceptance?
- Are these kinds of specifications adequate?
  - "The system interface shall be easy to use."
  - "The system shall support the maximum possible number of simultaneous users"

Specifying Quality Requirements

- When using natural language, write objectively verifiable requirements when possible
  - Load handling: "The system will support up to 100 concurrent users while while maintaining a response time under 15 ms."
  - Maintainability: "The following kinds of requirements changes will require changes in no more than one module of the system…"
  - Performance:
    - "System output X has a deadline of 5 ms from the triggering input event."
    - "System output Y must be updated at a frequency of no less than 20 ms."
- Provides unambiguous requirement even if it is not practical to test for compliance

Example Timing Requirements

5.2. TIMING REQUIREMENTS FOR DEMAND FUNCTIONS

For all the demand functions, the rate of demand is so low that it will not constitute a significant CPU-load.

For the starred entries, the desired maximum delay is not known; the entry is the maximum delay in the current OFP, which we will use as an approximation. In one case, both the current and desired values are given. The current value would be good enough to satisfy requirements, but the desired rate would be preferred.

Function name	Maximum delay to completion
IMS:	
Switch AUTOCAL light on/off	*200 ms
Switch computer control on/off	*200 ms
Issue computer failure	not significant
Change scale factor	*200 ms
Switch X slewing on/off	*200 ms
Switch Y slewing on/off	*200 ms
Switch Z slewing on/off	*200 ms
Change latitude-greater-than-70-degrees	*200 ms
Switch INA light on/off	*200 ms
FLR:	
Enable radar cursor	200 ms
Slave or release slave	40 ms

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
  - Using them to check the quality of the implemented system

Questions?

Midway Review: Expected Progress

- Look for progress against plan (limited evaluation of quality)
- Home page: name, logo, directory
- Plan and schedule:
  - Up-to-date plan: risks, process, etc.
  - Schedule tracking progress
  - Meeting notes and Developer Logs
- Requirements
  - Use cases to current iteration or beyond
  - Some detailed requirements
- Design: initial design (up to iteration)
- QA Plan: planned reviews and test cases

Grading Rubric

- For final deliverables: evaluate extent team *demonstrates* control?
- Managerial
  - How well does the team establish then follow a plan?
  - Does the team effectively use resources to 1) meet deadlines or 2) re-plan when needed?
- Intellectual
  - Does the team effectively establish what they intend to build?
    - To what extent is it consistent with customer?
  - Does the team build to the specifications?
  - How well does the team demonstrate correctness?
- Individual: to what extent did you contribute?