V22.0474-001 Software Engineering
Spring 2008

Lecture 4

Clark Barrett, New York University

Review

- The Rational Unified Process
- Extreme Programming

Outline

Software Requirements

- Requirements
- Requirements Engineering
- Team Assignments

Sources for today's lecture:

Necula, George and Aiken, Alex. *CS 169: Software Engineering*. Lecture notes, Fall 2004.

McConnell, Steve. *Code Complete*, Second Edition, ch. 3.

Sommerville, Ian. *Software Engineering*, Eighth Edition, ch. 6, 7.

Sommerville, Ian. Presentation slides for ch. 7 from http://www.cs.st-andrews.ac.uk/~ifs/Books/SE8/Presentations/.

Definitions

Requirements are the descriptions of the services provided by the system and its operational constraints.

Definitions

Requirements are the descriptions of the services provided by the system and its operational constraints.

The process of finding out, analyzing, documenting, and checking these services and constraints is called *requirements engineering (RE)*.

Definitions

Requirements are the descriptions of the services provided by the system and its operational constraints.

The process of finding out, analyzing, documenting, and checking these services and constraints is called *requirements engineering (RE)*.

Requirements Engineering

- Every software development process goes through a requirements engineering phase.
- It could be a simple as formulating in your mind what you want to program or as complex as a 500 page document produced over a period of several months.
- In this lecture we will discuss requirements and requirements engineering.

Why Requirements?

What are the advantages of a complete set of documented requirements?

Why Requirements?

What are the advantages of a complete set of documented requirements?

- Ensures that the user (not the developer) drives system functionality
- Helps avoid confusion and arguments about development process
- Helps minimize changes after development begins

Why Requirements?

What are the advantages of a complete set of documented requirements?

- Ensures that the user (not the developer) drives system functionality
- Helps avoid confusion and arguments about development process
- Helps minimize changes after development begins

Changes in requirements

- Requirements changes are expensive: changing the requirements costs
  - 3 times as much during the design phase
  - 5-10 times as much during implementation
  - 10-100 times as much after release $[CC2, p.39]$
- A careful requirements process doesn't mean there will be no changes later
  - Average project experiences about a 25 percent change in requirements
  - This accounts for 70-80 percent of the rework on the project $[CC2, p.40]$
  - Important to plan for requirements changes (more on this later)

Requirements

Unfortunately, the term *requirement* is not used in a consistent way in the software industry.

Requirements may be at different levels of abstraction

- *User requirements*
  - From the user's point of view
  - In natural language or diagrams
  - Describes the services and constraints of the system
- *System requirements*
  - From the developer's point of view
  - Should be precise and cover all the cases
  - Sets out the system's functions, services, and constraints in detail.
  - Also called a *functional specification*

Kinds of Requirements

System requirements can be roughly categorized into three categories

- *Functional system requirements*
  - Services the system should provide
  - What the system should or should not do in response to a given input
- *Non-functional system requirements*
  - Constraints on the services or functions offered by the system
  - Include timing constraints, constraints on the development process, standards
  - Often apply to the system as a whole
- *Domain requirements*
  - Requirements from the application domain of the system
  - May be functional or non-functional

Non-Functional requirements

Non-functional requirements can be further categorized into three areas.

- *Product requirements* Non-functional requirements on product behavior. May include requirements on properties such as timing, performance, memory, reliability, portability, usability, etc.
- *Organizational requirements* Requirements derived from policies and procedures in the customer's and developer's organization. May include process requirements, implementation requirements, delivery requirements, etc.
- *External requirements* Requirements derived from factors external to the system and its development process. Examples include interoperability with other systems, legislative requirements, ethical requirements, etc.

Specific categories are less important than understanding the principle: *requirements need to cover all relevant activities in the scope of the development project*.

When defining requirements, try to consider the problem from all possible angles.

User Requirements

User requirements are usually the first attempt to describe the desired functionality of the system.

Often suffer from deficiencies

- *Lack of clarity* Language may be ambiguous and imprecise
- *Requirements confusion* Functional requirements, non-functional requirements, system goals, and design information may not be clearly distinguished
- *Requirements amalgamation* Several requirements may be expressed together as a single requirement
- *Incomplete* Important cases may not be covered
- *Inconsistent* Requirements may contradict themselves

Part of the goal in refining user requirements to system requirements or a system specification is to eliminate these issues.

User Requirements

To minimize deficiencies in user requirements, some simple guidelines are helpful

- *Separate requirements* Each requirement should be clearly and separately identified. Functional and non-functional requirements should be separated.
- *Include a rationale for each requirement* Helps clarify reasoning behind requirements. Especially useful when evaluating potential changes to requirements.
- *Invent a standard format* Makes omissions less likely. Requirements are easier to check.
- *Distinguish between mandatory and desirable requirements*
- *Emphasize key parts of requirement* Use text highlighting or some other technique
- *Avoid technical jargon if possible* Keep user requirements simple

System Requirements

System requirements elaborate the user requirements to get a complete and precise description of the system.

Depending on the project and process, there may be no separation into user and system requirements. It may be helpful to think of user requirements as the early versions of system requirements.

An important consideration is the *notation* for system requirements.

Often, natural language is used.

System Requirements

Example

If sales for current month are below target sales, then report is to be printed unless difference between target sales and actual sales is less than half of difference between target sales and actual sales in previous month, or if difference between target sales and actual sales for the current month is under 5%.

What's wrong with this requirement?

System Requirements

Example

If sales for current month are below target sales, then report is to be printed unless difference between target sales and actual sales is less than half of difference between target sales and actual sales in previous month, or if difference between target sales and actual sales for the current month is under 5%.

What's wrong with this requirement?

- Hard to read
- Ambiguities: what are "sales", "target sales", 5% of what?
- Incomplete: what if sales are above target sales?

System Requirements: Notation

Natural language (informal) requirements

- Universally reviled: by academics, “how-to” authors, respectable pundits
- Also widely practiced. Why?

System Requirements: Notation

Natural language (informal) requirements

- Universally reviled: by academics, “how-to” authors, respectable pundits
- Also widely practiced. Why?
  - Customers, programmers, managers must read requirements
  - Least-common denominator effect
  - Finding a better notation is hard

System Requirements: Notation

Natural language (informal) requirements

- Universally reviled: by academics, “how-to” authors, respectable pundits
- Also widely practiced. Why?
  - Customers, programmers, managers must read requirements
  - Least-common denominator effect
  - Finding a better notation is hard

Alternatives to natural language:

- *Structured natural language* Only slightly better than natural language. Standard forms or templates are used to bring some rigor to unstructured natural language.
- *Graphical notations* Dependencies and functionality are modeled using boxes and arrows. Can be very helpful, but must be careful: boxes and arrows mean different things to different people (more on this later).
- *Mathematical specifications* Notations based on mathematical models of computation: logic, state machines, etc. Precise but time-consuming and hard for many people to understand.

System requirements: Notation

An analogy

Over many centuries, mathematics moved from natural language to a clear and concise algebraic notation:

System requirements: Notation

An analogy

Over many centuries, mathematics moved from natural language to a clear and concise algebraic notation:

Archimedes (ca. 225 BC)

> Any sphere is equal to four times the cone which has its base equal to the greatest circle in the sphere and its height equal to the radius of the sphere.

System requirements: Notation

An analogy

Over many centuries, mathematics moved from natural language to a clear and concise algebraic notation:

Archimedes (ca. 225 BC)

> Any sphere is equal to four times the cone which has its base equal to the greatest circle in the sphere and its height equal to the radius of the sphere.

Today

$$V = \frac{4}{3}\pi r^3$$

System requirements: Notation

An analogy

Over many centuries, mathematics moved from natural language to a clear and concise algebraic notation:

Archimedes (ca. 225 BC)

> Any sphere is equal to four times the cone which has its base equal to the greatest circle in the sphere and its height equal to the radius of the sphere.

Today

$$V = \frac{4}{3}\pi r^3$$

Discussion question: How is this bit of history relevant to software requirements notation?

System requirements: Notation

An analogy

Over many centuries, mathematics moved from natural language to a clear and concise algebraic notation:

Archimedes (ca. 225 BC)

> Any sphere is equal to four times the cone which has its base equal to the greatest circle in the sphere and its height equal to the radius of the sphere.

Today

$$V = \frac{4}{3}\pi r^3$$

Discussion question: How is this bit of history relevant to software requirements notation?

- Formal is better, but only if everyone understands
- It can take a long time to find the right notation
- Software requirements notation is still a work in progress

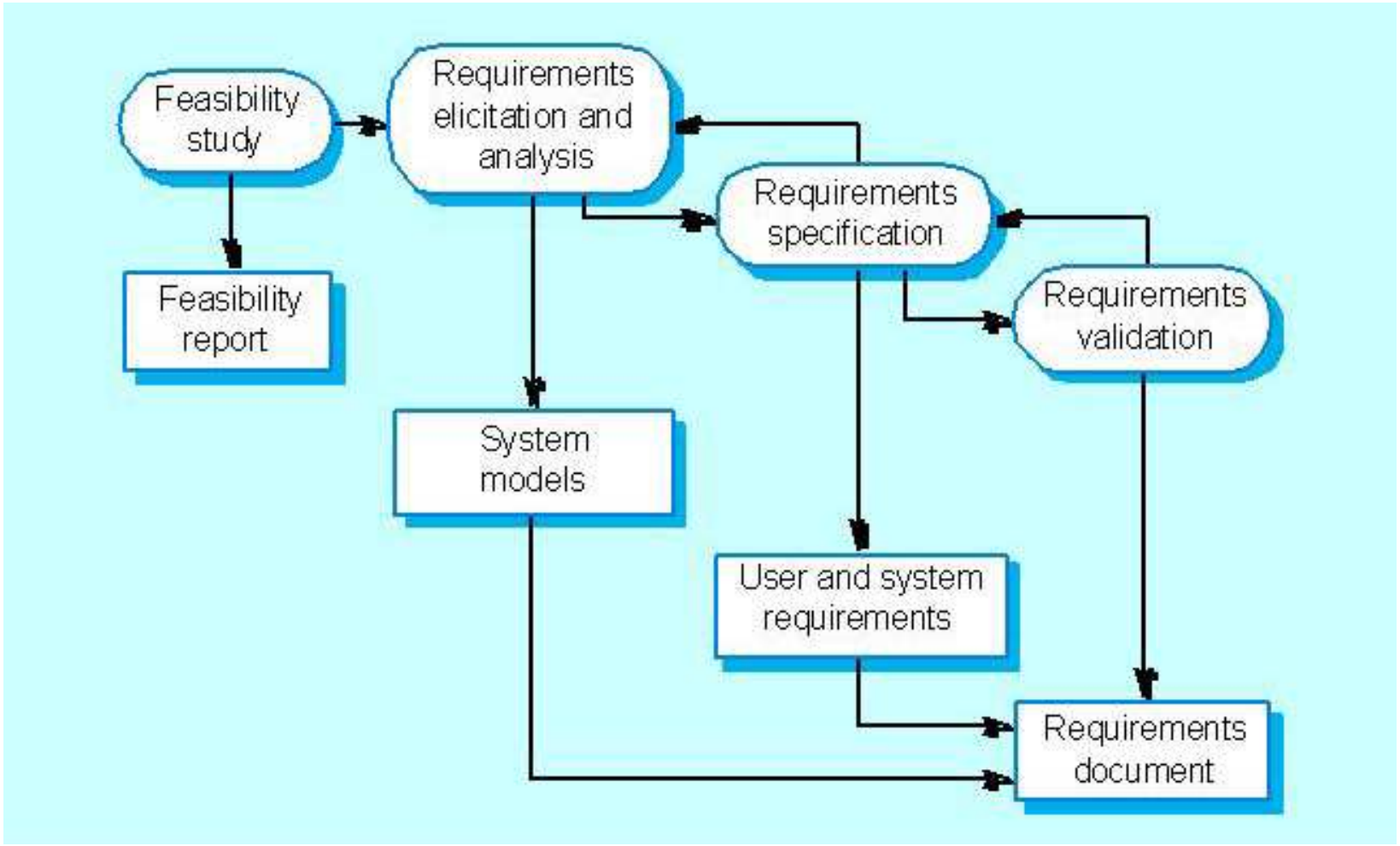
Requirements Engineering

Goal: Create and maintain system requirements.

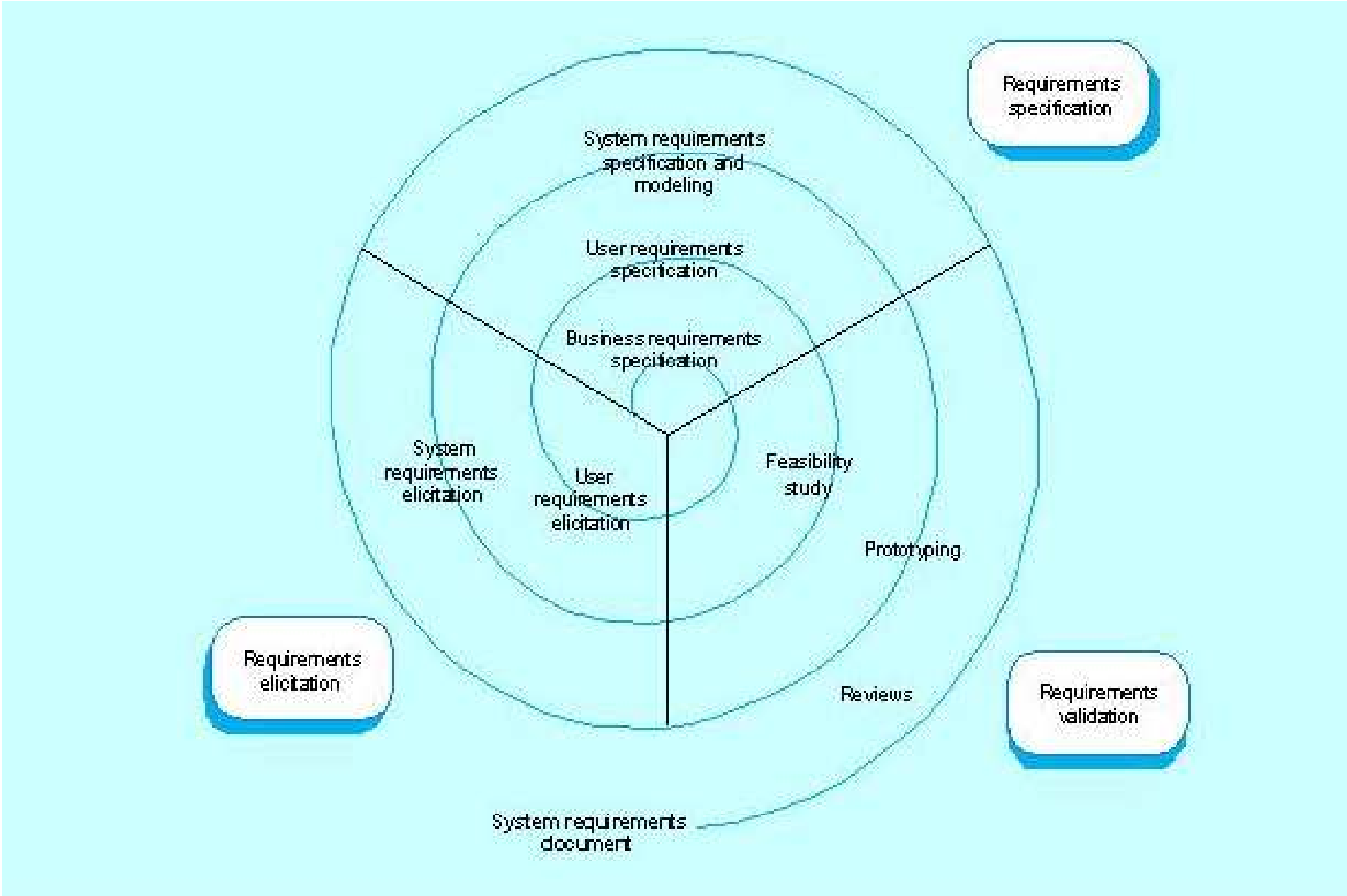
Four important activities

- Feasibility studies
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

Views of Requirements Engineering



Views of Requirements Engineering



Feasibility Studies

Before development begins on the system, it is often wise to do a *feasibility study*.

Answer questions such as:

- Does the system contribute to the overall objectives of the organization?
- Can the system be implemented using current technology and within given cost and schedule constraints?
- Can the system be integrated with other systems already in place?

Doing the study

- Consult information sources: managers, software engineers, technology experts, end-users
- Ask questions
  - How does the system help meet business objectives?
  - What would happen if system were not implemented?
  - What technology does the system require?
- Study should be quick (e.g. 2-3 weeks)

Requirements Elicitation and Analysis

A *stakeholder* is any person or group who will be affected by the system, directly or indirectly.

Elicitation of requirements should involve all stakeholders. This can be challenging because

- Stakeholders often don't know or may find it difficult to articulate what they want
- Stakeholders naturally express requirements in their own terms which may include implicit knowledge and domain-specific jargon
- Different stakeholders may have differing or even conflicting views
- Political factors may introduce inefficiencies or sub-optimal requirements
- In many cases, requirements are dynamic, changing with a changing economic and business environment

Requirements Elicitation and Analysis

Viewpoints are a way of classifying stakeholders and other sources of requirements.

Three categories of viewpoints

- *Interactor viewpoints* People or other systems that interact with the system
- *Indirect viewpoints* Stakeholders who do not use the system directly but who influence the requirements in some way
- *Domain viewpoints* Domain characteristics and constraints

More specific types of viewpoints

- Providers and receivers of system services
- Systems interfacing with the new system
- Regulations and standards applying to the system
- Sources of business and non-functional requirements
- Engineering viewpoints: developing, managing, maintaining system
- Marketing viewpoints

Requirements Elicitation and Analysis

Once viewpoints have been identified, they should be prioritized.

Priorities inform the important step of *requirement discovery*

Approaches to requirement discovery

- Interviewing
- Scenarios
- Use-cases

Requirements Elicitation and Analysis: Interviewing

Meetings with stakeholders

- Sit down together and ask questions
- Listen to what is said and unsaid
- Have a set of fixed questions
- Be prepared to ask flexible follow-up questions

Master-apprentice approach

- Have them teach you what they do
- Go to workplace and watch the process
- Learn requirements first-hand

Get Details

- Ask for copies of reports, logs, emails, etc.
- Details may support, complete, or contradict what the stakeholder said

Requirements Elicitation and Analysis: Scenarios

It is often easier for people to relate to real-life examples than abstract descriptions.

Scenarios describe a particular sequence of events that could take place within the proposed system.

Scenarios typically include:

- A description of what the system and users expect when the scenario starts
- A description of the normal flow of events in the scenario
- A description of what can go wrong and how this is handled
- Information about other activities that might be going on concurrently
- A description of the state of the system when the scenario finishes

Requirements Elicitation and Analysis: Use-Cases

Use-cases are a scenario-based technique for describing requirements.

Use-cases

- identify the type of interaction and the actors involved
- may describe a particular scenario or a set of possible scenarios
- are typically illustrated with UML or similar diagrams (more on this later)
- are most effective at capturing *functional* requirements

A use-case based requirements methodology must take care to cover all possible interactions within the system.

Requirements Validation

Once the requirements have been captured in a document, they should be checked in a number of different ways.

Important requirements checks

- *Validity checks* Make sure that the captured requirements are really what the stakeholders want and need (i.e. get stakeholder feedback).
- *Consistency checks* Requirements in the document should not conflict.
- *Completeness checks* The requirements should cover all possible situations and behaviors for the system.
- *Realism checks* The requirements should be double-checked for feasibility, taking into account the budget and schedule.
- *Verifiability* Requirements should be verifiable. There should be a way to demonstrate that the system meets its requirements.

Requirements Validation

Techniques for requirements validation

- *Requirements reviews* A team which includes representative stakeholders manually checks requirements.
- *Prototyping* An executable model is built based on the requirements. Feedback is solicited from stakeholders
- *Test-case generation* Requirements are converted into specific test cases. This is the approach taken by XP.

Requirements Management

Handling requirements changes

- *Make sure you understand requirements before moving on to design* Good design will not compensate for bad requirements. You will have to do it over again when the requirements are fixed.
- *Make sure everyone knows the cost and benefit of requirements changes* Clients get excited about new features. They need to be reminded that this will impact the cost and schedule. Changes that do not fit in with the business case should be rejected.
- *Set up a change-control procedure* Plan for changes. Make sure all impacted stakeholders have a say in proposed changes.
- *Use the right software process* Evolutionary approaches more easily accommodate frequent changes.

Requirements

Summary

- Requirements are about understanding what you are going to do before you do it
- A better understanding means less likelihood of changes which means a better product and faster time-to-market
- All stakeholders should be involved
- Requirements should be as clear and precise as possible, while remaining understandable by those affected by the requirements
- It is important to validate requirements
- It is important to plan for changes in requirements

Teams

Exercise and Nutrition Log

- Rachal, Andy, Simba

Assault on the Exploding Barrel Factory

- Jack, Ben, David

Project

This is a real project

- I expect you to work to build a real system
- To be used by real people
- No play acting

Take responsibility

- You are expected to find and solve problems
- Coding is only part of the job
- Be prepared for a lot of other work

Come to me for advice if you get stuck

Project

Teams have been assigned

- This is how it is in the real world
- I took into consideration preferences and experience

Statistics on team assignments

- 4 people got their first choice, 2 second choice
- Most partner requests were honored

Potential Team Problems

Today you step into the unknown. To have a chance of returning successfully, your team must function well together.

People problems often trump technical issues: this is often the reason projects fail.

Different goals

- Do a great project
- Get a good grade
- Just survive
- Differences in goals will dog every step

Team members don't keep their commitments

- Makes planning very difficult
- Can't give "unreliable" members anything significant

Avoiding and Overcoming Team Problems

Communicate!

- Early and often, among yourselves
- Schedule a group meeting right away
- Schedule at least weekly meetings throughout the semester

Talk about Goals

- Work to make and keep commitments to shared goals
- Don’t make a commitment you can’t keep
- Talk about missed deadlines: don’t ignore problems and hope they go away

Be a team, not just a group!

- Set tone early: team culture is like cement (sets quickly, hard to alter later)
- Everyone needs to take responsibility
- Remember that your teammates will rate your performance at the end of the semester
- Have a high standard, but remember that no one is perfect

Homework: Requirements Document

Requirements Document and Presentation

- Your first team assignment is a Requirements Document
- Details are posted on the web page
- Division of labor is your responsibility
- Team gets one grade on team assignments
- Requirements document is due on Feb. 12
- Class presentations are on Feb. 14 (more on this next time)

Goals

- Flesh out project concept
- Fix weaknesses
- OK for project to morph from original idea
- Must have an external customer
- Talk to customer to gather requirements