# Lecture 5: Requirements

CS 5150, Spring 2025

# Lecture goals

1. Understand and Document verifiable requirements
2. Elicit requirements from stakeholders

# Administrative Reminders

- Teams have been formed (mostly)
  - Should have selected team Client
  - Fill up the survey if you still need a team
- Project plan due Fri (Feb 7)
- Schedule meeting with your client (Client info on canvas)
- For external client (Beware): Please try not to be too ambitious!

# Course Grading

- **Project**: 60%
  - Reports, Presentations, Code/Process quality, Client feedback, Peer Evaluations
- **Assignments**: 20%
  - 4-5 assignments
- **Attendance/Participation**: 10%
  - Attend 75% of classes, engage in class
- **Exams**: 10%
  - Two in-class exams

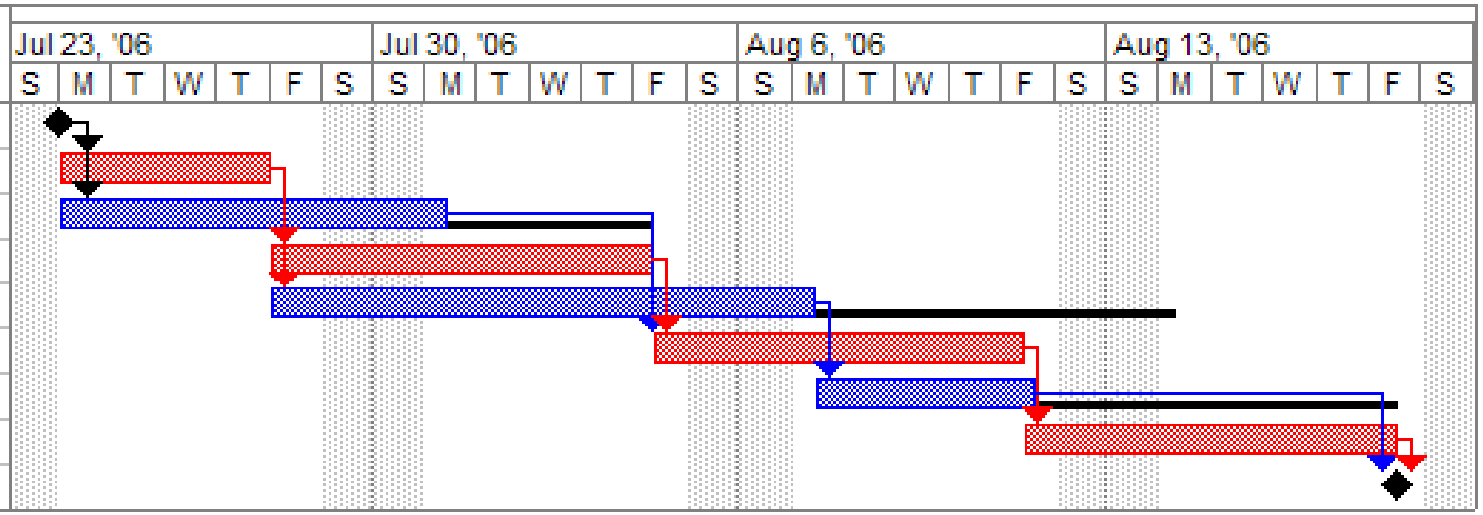
# Project plans

… continued from Lecture 4

# Gantt charts

- Visualize plan for when activities actually take place
  - Number of parallel activities limited by resources
- Can highlight critical path, slack time

| ID | Task Name | Predecessors | Duration |
|---|---|---|---|
| 1 | Start | | 0 days |
| 2 | a | 1 | 4 days |
| 3 | b | 1 | 5.33 days |
| 4 | c | 2 | 5.17 days |
| 5 | d | 2 | 6.33 days |
| 6 | e | 3,4 | 5.17 days |
| 7 | f | 5 | 4.5 days |
| 8 | g | 6 | 5.17 days |
| 9 | Finish | 7,8 | 0 days |

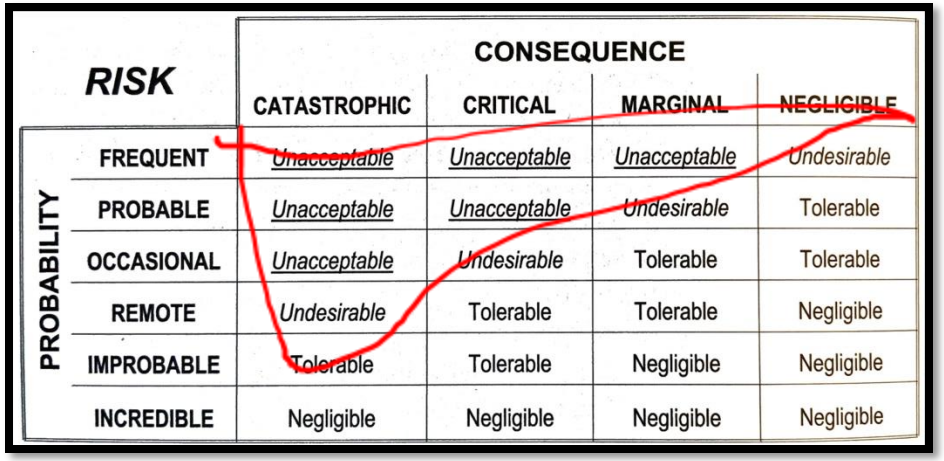




CC-By-SA Dbsheajr at English Wikipedia

# Risk management

1. Identify risks
   - Brainstorm what could go wrong
2. Analyze risks
   - Determine likelihood and consequence
   - Prioritize based on "risk"
3. Plan
   - Avoidance: reduce likelihood
   - Mitigation: reduce consequence
   - Contingency: "Plan B"
4. Monitor
   - Update risks regularly



| RISK | CONSEQUENCE | | | |
|---|---|---|---|---|
| PROBABILITY | CATASTROPHIC | CRITICAL | MARGINAL | NEGLIGIBLE |
| FREQUENT | Unacceptable | Unacceptable | Unacceptable | Undesirable |
| PROBABLE | Unacceptable | Unacceptable | Undesirable | Tolerable |
| OCCASIONAL | Unacceptable | Undesirable | Tolerable | Tolerable |
| REMOTE | Undesirable | Tolerable | Tolerable | Negligible |
| IMPROBABLE | Tolerable | Tolerable | Negligible | Negligible |
| INCREDIBLE | Negligible | Negligible | Negligible | Negligible |

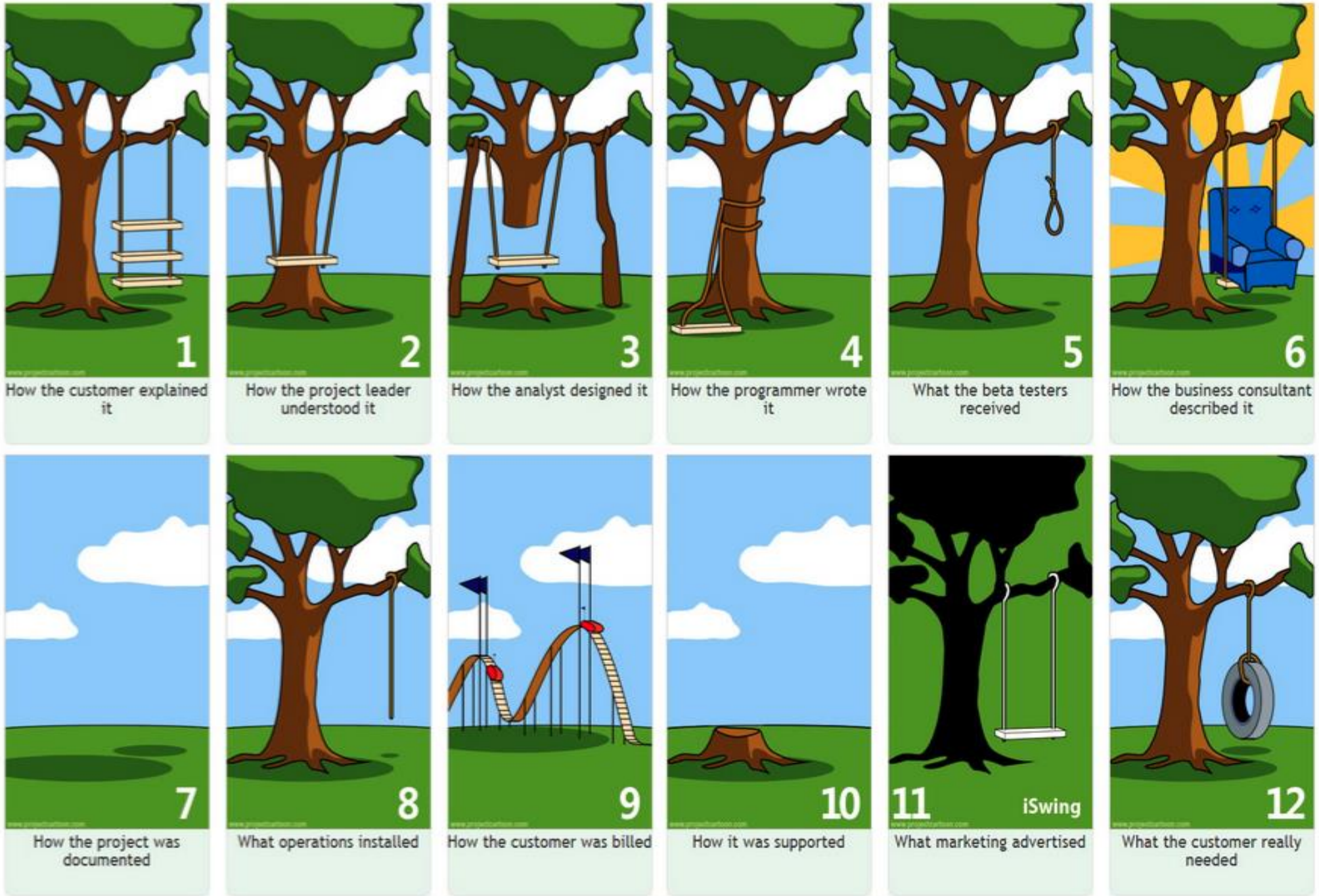


*Better Embedded System Software*. Koopman 2010

# CS 5150 project plans

- Feasibility studies and project plans should be written
  - Well-written, well-presented – review entire document
  - Short enough that everybody reads it
  - Long enough that no important topics are skipped
    - A report that is not read and understood is not useful
- Keep in mind:
  - Team availability, team skills
  - Time constraints
  - Equipment and software
  - Start-up time
  - Client availability
  - Scope (not vague, not too ambitious)

# Requirements


1
How the customer explained it
2
How the project leader understood it
3
How the analyst designed it
4
How the programmer wrote it
5
What the beta testers received
6
How the business consultant described it
7
How the project was documented
8
What operations installed
9
How the customer was billed
10
How it was supported
11
iSwing
What marketing advertised
12
What the customer really needed

# Requirements: Purpose

- What should a product do?
- What should a product *not* do?
- How is a product constrained?
- Take client's perspective
  - Meeting requirements should provide meaningful visibility
  - Not about design – "what", not "how"
- How should a product be tested?

- Risks of insufficient requirements documentation
  - Client dissatisfaction
  - Late discovery/rework
  - Poor design tradeoffs

- *Code is not a specification*

# Top reasons for project failure

| | |
|---|---|
| **Incomplete requirements** | 13.10% |
| **Lack of user involvement** | 12.40% |
| Lack of resources | 10.60% |
| **Unrealistic expectations** | 9.90% |
| Lack of executive support | 9.30% |
| **Changing requirements & specifications** | 8.80% |
| Lack of planning | 8.10% |
| **System no longer needed** | 7.50% |

- Failure to understand the requirements led developers to build the wrong system

*The CHAOS Report (1994)*, The Standish Group

# Subphases

1. **Analysis**: Establish functionality in consultation with stakeholders
2. **Modeling**: Organize requirements systematically
3. **Definition/Specification**: Record and communicate precise requirements

# Heavyweight vs. Lightweight

**Heavyweight**

- Gather most requirements upfront
- Document requirements formally

**Lightweight**

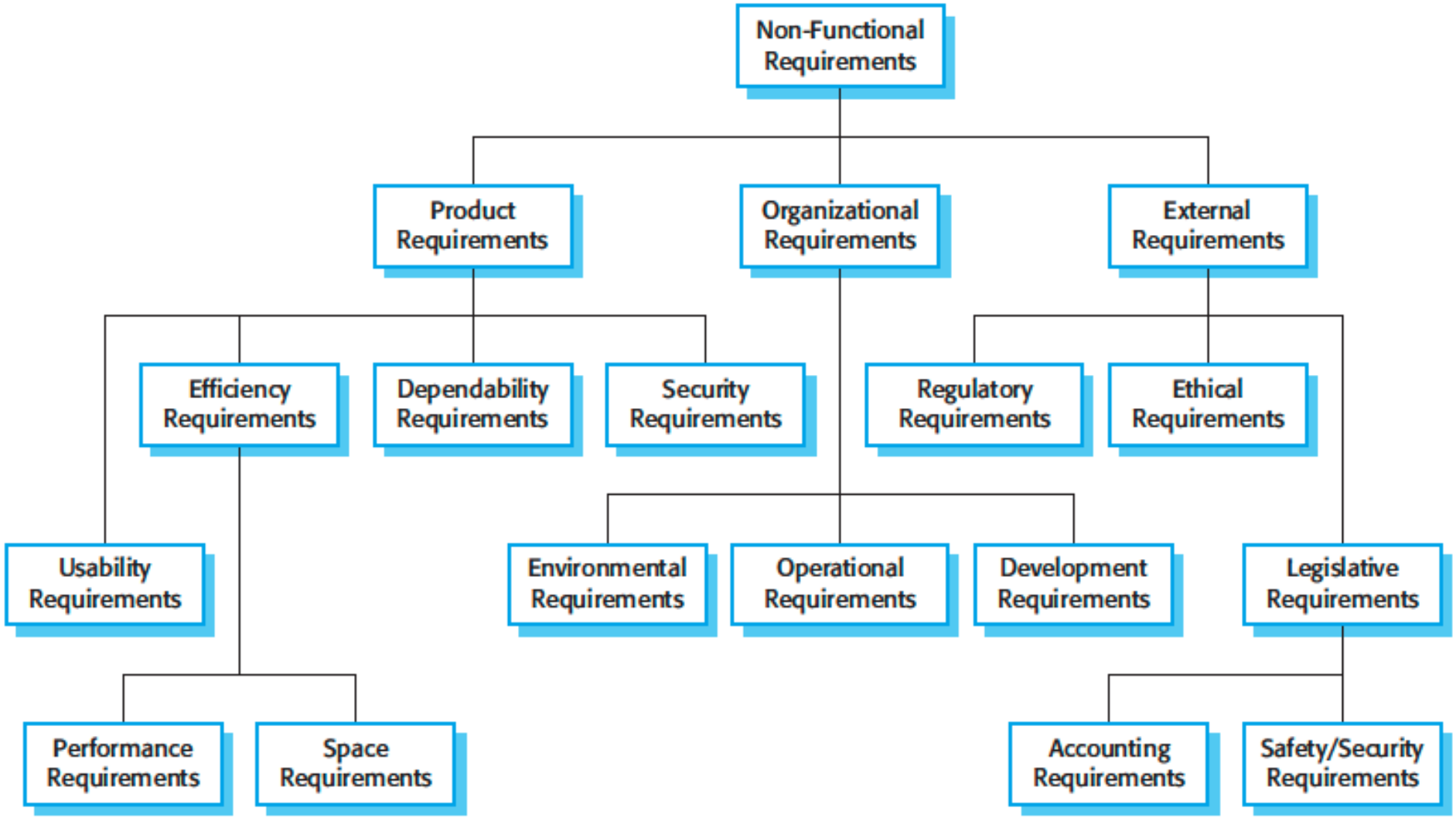
- Start with system-level requirements
- Expand and refine requirements iteratively (e.g., for each sprint)
  - Continual client interaction

*Requirement still exist and should still be documented*

# Types of Requirements

- **Functional**
  - What a product should do
  - What a product should not do
  - Can be verified locally
- **Non-functional**
  - Aka "quality requirements"
  - Property of system as a whole
- **Constraints**
  - Limits how the system can be built

- Examples:
  - "When a document link is visited, it shall display the document **only if** the user is authorized to read it; otherwise, it shall display a permissions error."
  - "Visual feedback from tapping a control shall be displayed **within 100ms** of contact."
  - "Records of queries issued by users shall be stored in an **Oracle** database."


Non-Functional Requirements
Product Requirements
Organizational Requirements
External Requirements
Efficiency Requirements
Dependability Requirements
Security Requirements
Regulatory Requirements
Ethical Requirements
Usability Requirements
Environmental Requirements
Operational Requirements
Development Requirements
Legislative Requirements
Performance Requirements
Space Requirements
Accounting Requirements
Safety/Security Requirements

# Exercise: Refining informal requirements

- "Customers should be able to enjoy the game on their laptop"

How to refine these requirements?

Power requirements (<= XY Watts)
GPU requirements (max 12GB GPU VRAM)
Different controllers
30fps

# Validation & Verification

**Validation**

- "Are you building the right thing?"
  - Would a system satisfying all of the requirements (and nothing else) meet the business need?
  - Are assumptions in models consistent with reality?
- Involve client
  - User testing
  - Acceptance testing

**Verification**

- "Did you build it right?"
  - Implementations should be verified against requirements
  - Design can be verified by analysis
  - Process can be verified by audits
- Testing
  - Can define pass/fail criteria based on previous step

# Requirements Definition

- Audience: Client AND developers

- CS 5150: Use future report/presentation to validate requirements with client
  - "Our understanding of your requirements is that …"

# Writing good requirements

- Must be verifiable
  - Can it be measured?
    - Use proxy measurements if needed
  - Are tolerances specified?
  - Can you design a test?
    - Include pass/fail criteria
  - Is it feasible? (to implement AND to verify)
- Must relate to **client-relevant** behavior
- Use consistent wording
  - "Shall"
  - "Should" if there are exceptions
  - Consistent names for actors, interactions, events
- Use appropriate format
  - Flow chart, decision table, ...
- Provide rationale
  - Link to requirements being derived from or depended on

# Poll: Is this a good requirement?

*When the timer expires, the software shall increment 16-bit integer variable `rollOverCount`.*

PollEv.com/cs5150sp25

# Improvement?

*The system shall keep a count of how many timer expirations have occurred, with the ability to tally at least 15,000 expirations.*

# Realistic tolerances

"The game shall render 30 frames each second on a Nintendo Switch."

# Tracking and tracing

**Objective**: facilitate verification, validation, revision

- Complete list
- Unique identifier
- Organized, cross-linked
- Linked to verification activities
  - Separate document (e.g., verification matrix)
- Change review procedure

| Runs/Tests | T1 | T2 | T3 | T4 |
|---|---|---|---|---|
| R1 | X | X | X | |
| R2 | X | X | | |
| R3 | X | | X | |
| R4 | | | | |

# Analysis

- Check for
  - Completeness
  - Consistency

- Example:
  1. Telemetry shall be transmitted every 30 minutes.
  2. The radio amplifier shall be powered off when <30% of battery charge remains.
- Example:
  1. When a calendar is marked "private," its appointments shall not be visible to other users.
  2. When booking a meeting, the interface shall suggest time slots during which all invited attendees are available.

# Stakeholder & Viewpoint analysis

- Identify who is affected by the system (Viewpoints)
  - Client
  - Customers
  - Users (many categories)
  - Administrators
  - Maintainers
- Effort often not proportional to utilization
  - E.g., administrative capabilities are often much larger than user capabilities

# Brainstorm: Viewpoints for university admissions system

# Eliciting requirements

# Interviews

- Difficult, but essential
- Tips:
  - Allow plenty of time
  - Prepare before meeting client
  - Keep full notes
  - Clarify what you do not understand
    - Define domain-specific terminology
  - Repeat what you hear
- Consider all stakeholders
- Ask questions
  - "Why do you do things this way?"
  - "Is this essential?"
    - Be wary – impact may not be obvious
  - "What are the alternatives?"

# Negotiation and Prioritization

- Conflicts, and difficulties affecting cost and schedule, must be resolved with client
  - Help client understand the tradeoffs
  - Be open to suggestions
- Incremental delivery (e.g., Agile sprints) encourages regular prioritization

# Stories & scenarios

- Don’t start with formal specifications
  - Most clients can't relate to them
  - Difficult to evaluate completeness
- **Stories** put devs, client on same wavelength
  - Describe actors and their goals
  - High-level, "big picture"
  - Lavish detail about context
    - Helps crystalize alternative viewpoints
    - Refocus by asking which details are relevant
- **Scenarios** detail interactions with system
  - Agile user stories - narrative scenarios with moderate detail
    - Often written on cards
    - Devs break into tasks to estimate effort
    - Prioritized by clients for inclusion in a sprint
    - Postponed stories may be revised with minimal rework
  - Structured scenarios provide more detail
    - Tool for clarifying requirements, checking completeness

# Usage scenarios (or Stories)

- Illustrates some interaction with a proposed system
- Use specific examples from a user's point of view
- Clarifies many functional requirements
- Especially good for analyzing off-nominal behavior

- Must include:
  - Purpose
  - User or transaction being followed
  - Assumptions about equipment
  - Steps of scenario
- Should consider (corner cases)
  - What could go wrong
  - Concurrent activities
  - Changes to system state
- Avoid system details that pertain to design

# Developing scenarios with clients

- Choose a viewpoint
- Identify purpose,
  actors, equipment, procedure
- Ask clarifying questions

- Example: online exam system

# Online exam system: Viewpoints?

# Online exam system scenario: typical student

- Purpose: Describe how a typical student uses the system to take an exam.
- User:
- Equipment:
- Steps:

# Modeling requirements

# Requirements Modeling

- Need to bridge requirements and design
  - Leverage abstraction
  - Exploit patterns
  - Identify invariants
  - Improve precision

- UML
- Use cases
- Activity and flow diagrams
- State charts
- …

- Future lecture

Read Ian Sommerville's book: Chapter 4 and 5

# Requirements steps

1. Elicitation & Analysis
2. Modeling
3. Specification

- Heavyweight
  - Document formal specification before beginning design
- Lightweight
  - Relevant requirements developed during sprints
    - But work out system-level requirements upfront
  - Avoid specification unless necessary
    - Models, prototypes clearer to client
    - Sometimes details are important