Software Requirements

- Attributes and functionality of the software *from the user’s perspective* → what the software does
- Also should include *hardware and software constraints*
- In general, should NOT contain implementation details
Plan-Driven vs. Agile Requirements

- **Plan-driven → Software Requirements Specification (SRS)**
  - **Contract** between developers and customer
  - Specific, **formal** document
  - Defined **before work starts**
  - Best when requirements:
    - Largely determinable in advance
    - Mission-critical
    - Have low change rate
    - Affect many features at once
    - Quality or nonfunctional

- **Agile → often User Stories**
  - Written by customer → should not be too technical
  - Informal → often on **index cards** or in **spreadsheet**
    - **Strict**er format: “As a (role) I want (something) so that (benefit).”
    - Can also include: priority, time estimate, identifier
    - **Product Backlog** = priority sorted features/functions/tech enhancements
  - Stories may be modified, added, or removed as development proceeds
  - Best when requirements may **change considerably**
Why Have Official Requirements at All?

- Explicit requirements important because:
  - User (not programmer) drives system’s functionality
  - Avoids arguments and conflict
  - Minimizes changes to system after development begins
Dealing with Changing Requirements

- Review (changed) requirements and ensure they are good enough before continuing
- Make sure everyone understands cost of requirement changes
- Set up change-control procedure
- Use development approaches that accommodate changes
- Unworkable/unresolvable requirements $\rightarrow$ dump the project
- Keep you eye on the business case for the project

- **SRS should address:**
  - Functionality
  - External interfaces
  - Performance
  - Attributes
    - E.g., portability, correctness, maintainability, security, etc.
  - Design constraints imposed on an implementation

- **SRS on design:**
  - Limits range of valid designs
  - Does not specify PARTICULAR design
IEEE Std 830-1998: Characteristics of Good SRS

- An SRS should be:
  - Correct
  - Unambiguous
  - Complete
  - Consistent
  - Ranked for importance and/or stability
    - Clears out hidden assumptions
    - Properly prioritize effort and time
  - Verifiable
  - Modifiable
    - Good structure
    - Coherent cross-referencing
    - Avoid redundancy
    - Don’t intermix requirements
    - Less detail in high-level sections
  - Traceable
    - Backward traceability → why we have a requirement
    - Forward traceability → which requirement does this satisfy?
      - Each requirement should have unique name/reference number
IEEE Std 830-1998: SRS Outline

- Table of Contents
  1. Introduction → background for product
     o Purpose of SRS
     o Scope → identify products, explain functionality, talk about benefits/objectives/goals
     o Definitions, acronyms, and abbreviations
     o References
     o Overview of SRS organization
  2. Overall description → background for requirements
     o Product perspective → interfaces, operations, site adaptation
     o Product functions
     o User characteristics
     o Constraints
     o Assumptions and dependencies
     o Apportioning of requirements → requirements that may be delayed until future versions
  3. Specific Requirements
     o Specify (input) (operation on input) (output)
     o Externally perceivable
     o Uniquely identified
     o Includes: External Interfaces, Functions, Performance requirements, Logical database requirements, Design constraints, Software system attributes

- Appendixes
- Index
- **System mode**
  - E.g., In-game, in-game-menu, main-menu, level selection
- **User class**
  - E.g., elevator control system → passengers, maintenance workers, fire fighters
- **Objects**
  - Real-world entities modeled by system
- **Feature**
  - E.g., telephone system → local call, call forwarding, conference call
- **Stimulus**
  - E.g., aircraft landing system → loss of power, wind shear, etc.
- **Response**
  - E.g., all functions associated with generating list of current employees
- **Functional hierarchy**
  - Organized by common inputs, common outputs, or common internal data

- May use multiple levels of organization (system mode, then objects)
- May use diagrams, state charts, etc. where appropriate