Lecture 3: Requirements

CS 5150, Spring 2026

Administrative Reminders

- Submit project pitch (everyone must submit)
  - **New**: Indicate if you need additional members; any specific skills you are looking for
  - Post on Ed; See Team Matching Survey
- **Next**:
  - Course staff will review pitches and assign team members (by this week)
  - First Project Deliverable: Project Plan (Due Feb 5, EOD) – In class activity!
- Assignment 1 is out (Counting Lines of Code) – Due Feb 10, EOD

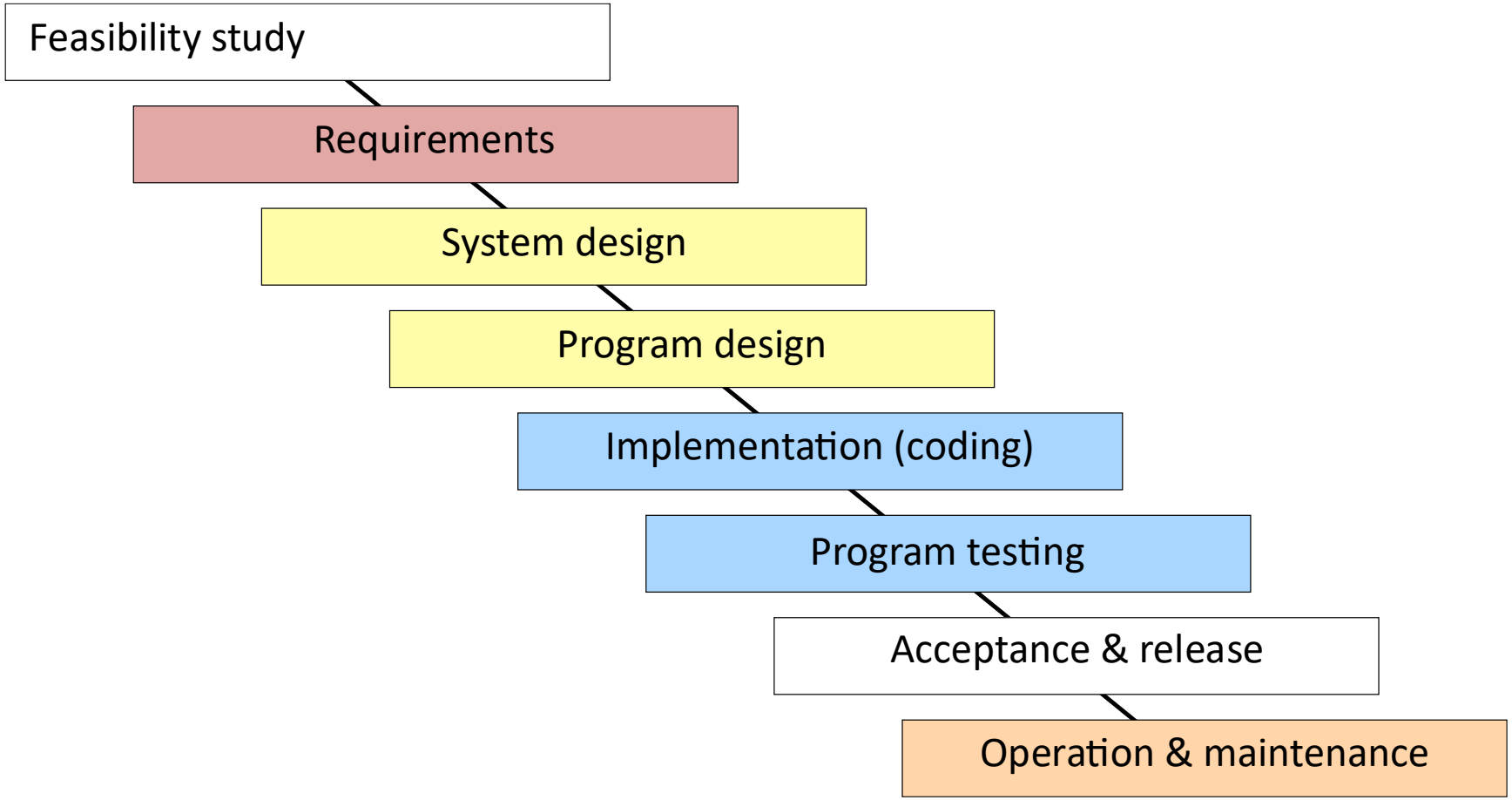
Projects & Processes

… continued from Lecture 2

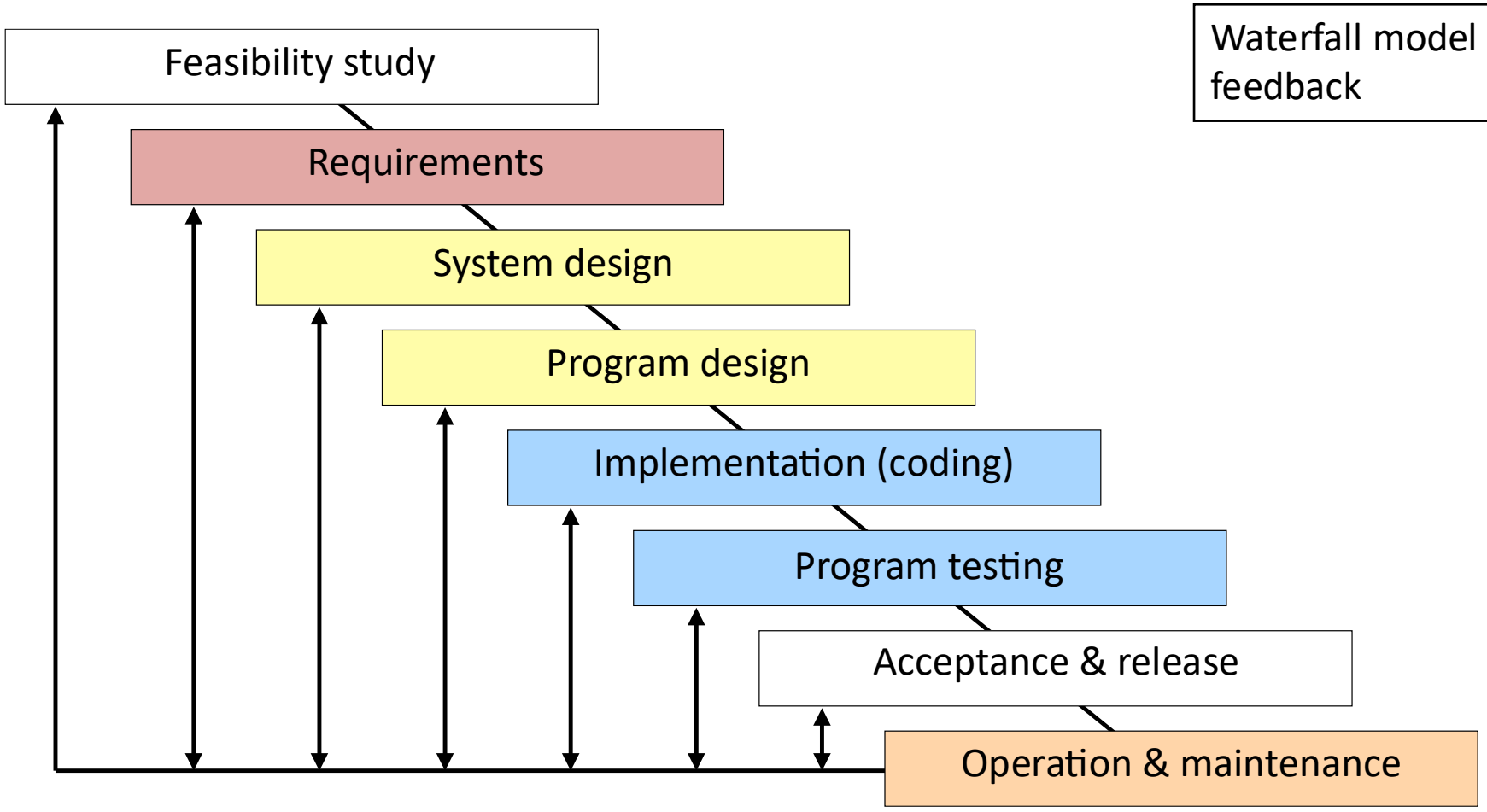
Previously…

- Stakeholders
- Risk, Minimization
- Software Development Processes/Software Development Life Cycle (SDLC)
  - Waterfall
  - Modified Waterfall
  - Iterative Refinement/Prototyping
  - Incremental Delivery
  - Agile Methods
  - COTS
  - Mixed Processes

The waterfall model



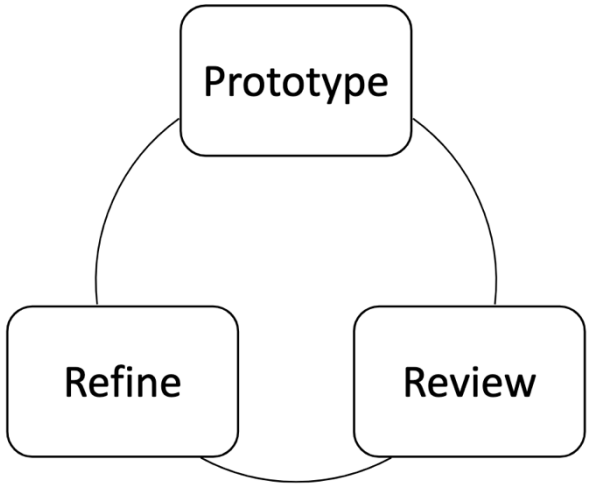
Modified waterfall model



Waterfall model with feedback

Iterative refinement: Prototyping

- Requirements are hard to elicit without an operational system
  - Especially for user interfaces
- Developers can learn a lot about the domain and proposed design through prototyping
- Process:
  - Create a prototype early on
  - Review prototype with clients; test prototype with users
  - Clarify requirements, improve design (revise documentation)
  - Refine prototype iteratively
- Prototype is not a releasable product!
  - Cannot evaluate non-functional requirements without final system design



When does it work well?

Incremental delivery

- Deliver fully-tested increments with subset of functionality
  - Start with a base system that matches final architecture, but with dummy components/missing functionality
  - Develop new components along with their test cases in isolation; when functional, add to base system
  - System is periodically built and tested to catch regressions
- Challenges:
  - Requires base system with good design, automated testing infrastructure (high startup overhead)
  - Code structure can degrade over time (refactoring is not a new component)
  - Increments have incomplete functionality (difficult to evaluate)

Examples in industry and govt

- SpaceX: Uses incremental development with spacecraft, launch vehicles, electronics and avionics, and operational flight hardware operations
- [1]: " … *SpaceX followed an iterative design process, continually improving prototypes in response to testing. Traditional product management calls for a robust plan executed to completion, a recipe for cost overruns …* "
- Government agencies: Vulcan rocket [2]

[1] https://qz.com/281619/what-it-took-for-elon-musks-spacex-to-disrupt-boeing-leapfrog-nasa-and-become-a-serious-space-company
[2] https://spacenews.com/evolution-of-a-plan-ula-execs-spell-out-logic-behind-vulcan-design-choices/

Agile models

- What is Agile all about?
- **Premise**: the world is uncertain, and we must be flexible and responsive to changes
- “Agile software development” is a general term for frameworks and practices outlined in **the Agile Manifesto**

https://agilemanifesto.org/principles.html

Agile Manifesto

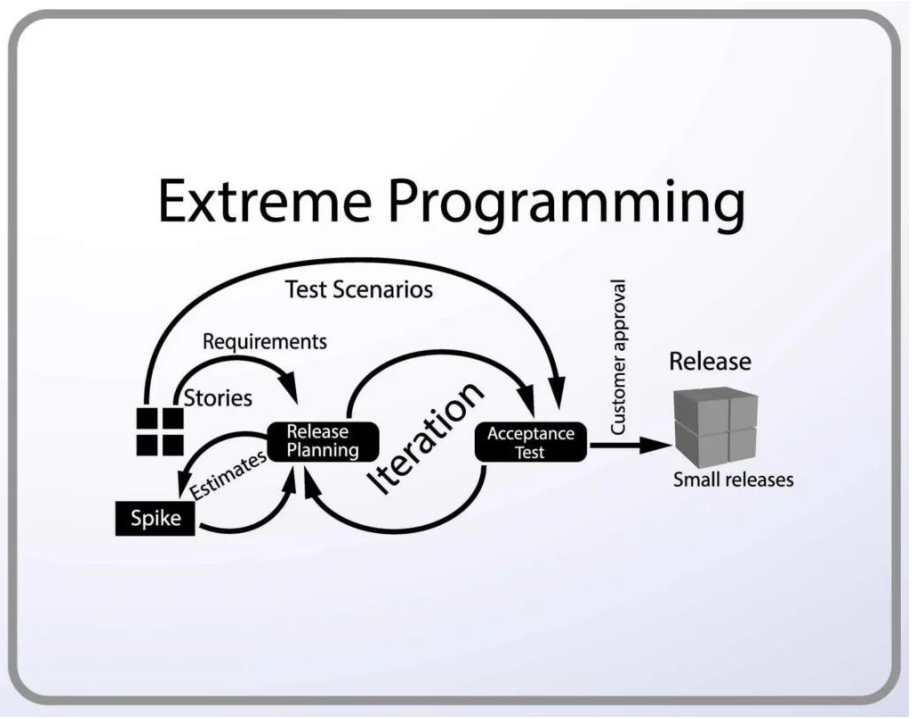
- **Individuals and interactions** *over* processes and tools
- **Working software** *over* comprehensive documentation
- **Customer collaboration** *over* contract negotiation
- **Responding to change** *over* following a plan

Agile methods (eXtreme Programming)

- User stories
  - Improves communication
- Incremental planning
- Small releases
  - Improves visibility
- Simple design
- Test-first development
  - Shifts left
- Periodic refactoring
- Pair programming
  - Shifts left
- Collective ownership
- Continuous integration
  - Shifts left
- On-site customer
  - Improves communication

Extreme Programming (XP)

- XP emphasizes how engineers should work – good practices taken to an extreme
- Examples:
  - Continuous testing and integration
  - 10-minute build
  - Constant discussions with customers
  - Full flexibility to change requirements anytime
  - Pair programming
  - Test-driven development



XP Practice: User Stories

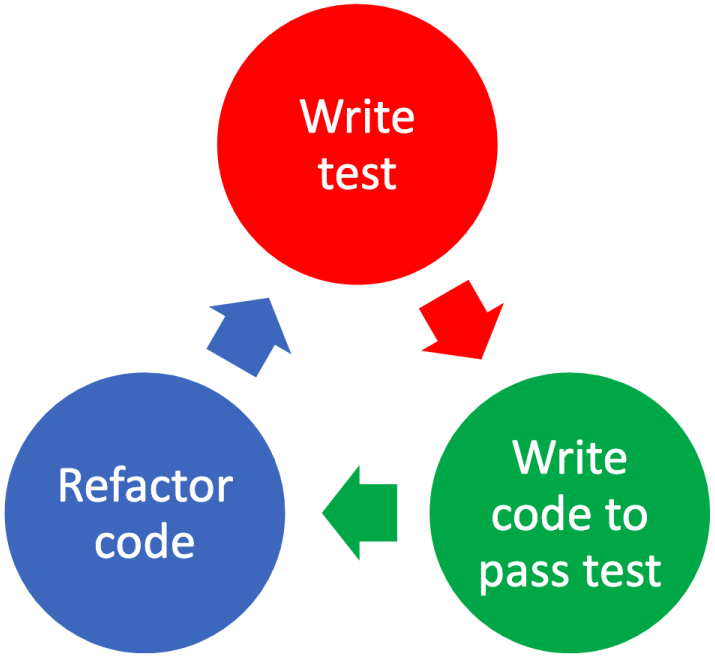
- Concise, user-focused descriptions of desired outcomes
- Helps understand “why” and breakdown large projects into manageable tasks
- Different from “requirements.”
- Example: “As a **project manager**, I **want** to generate a project status report **so that I can** share progress with stakeholders.”

XP Practice: Pair programming

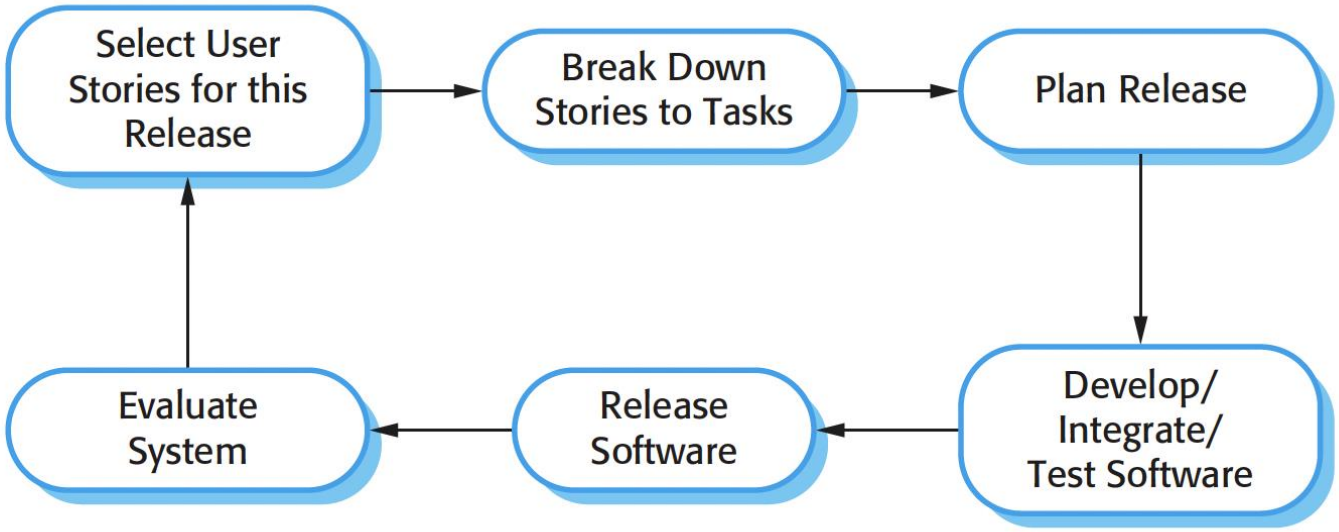
- Pair programming – All production software is developed by two people sitting at the same machine.
- Pairs and roles (driver/navigator) are frequently changed.
- Provides for continuous code development, collaboration, and review.

XP Practice: Test-driven development

- Start with requirements
- Write tests before code
- Develop code to make the tests pass
- Tests run early and often



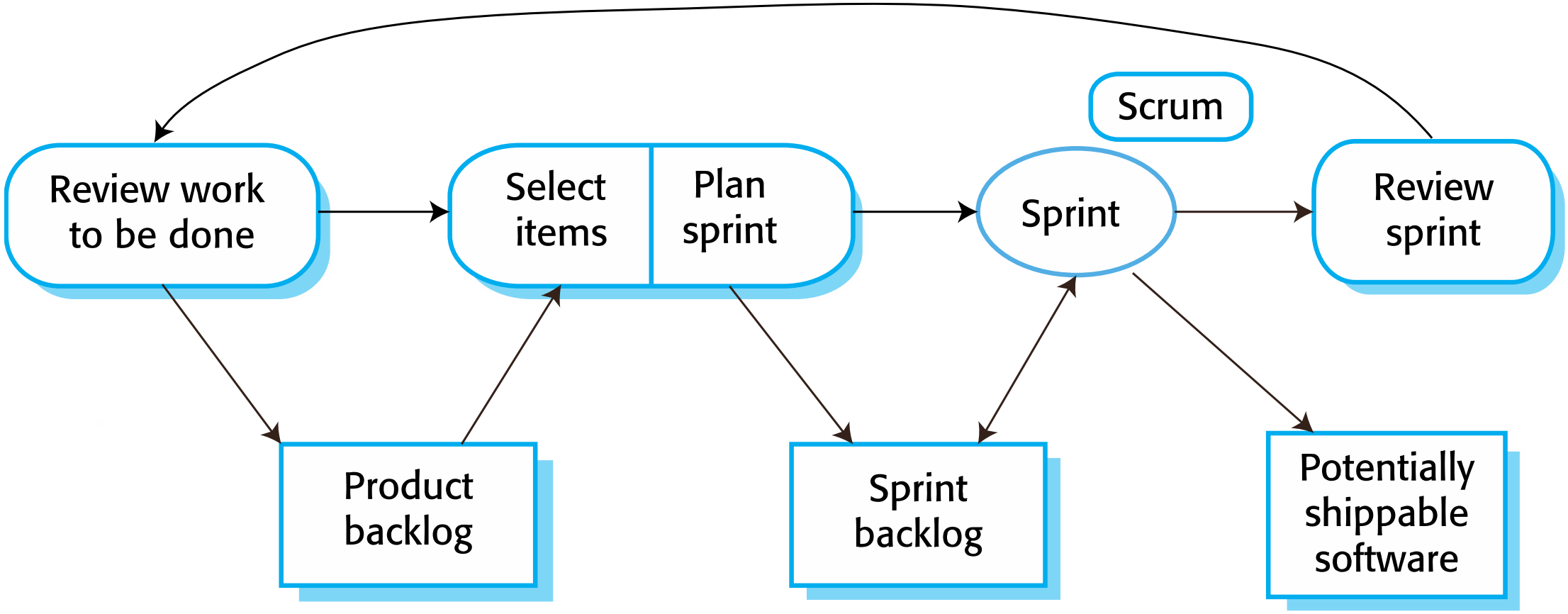
Extreme Programming



Scrum implementation of Agile

- Provides management structure that accommodates XP/Agile
- Work scheduled as "time boxes" (sprints)
  - 2-4 weeks
- Tasks selected from backlog
  - Incomplete work is *not* automatically carried over
- Sprint product is released, production-quality code + docs
  - Sprint planning defines an MVP
- Daily team meetings

Agile/scrum workflow



Agile/scrum

Benefits

Good visibility and communication

Accommodates change, fuzzy requirements

Very popular today for small, dynamic projects

Challenges

Tricky to scale to large organizations

Works best with highly-skilled, autonomous devs

Hard to validate requirements

Lack of formal docs impedes maintainence, handoff

Integration and configuration

- When system design is *standardized*, can better take advantage of **code reuse**
- Providers collect lots of configurable components into commercial-off-the-shelf (COTS) products
  - E.g. Enterprise Resource Planning (ERP) platforms
- Developers integrate, configure components based on client requirements
  - Effectively skip system design and program development steps

Pros
- Reduced cost and time

Cons
- Reduced function

Mixed processes

Many projects mix elements of multiple methodologies

- If requirements are well-understood, might use Waterfall to define requirements & system design, then implement using Incremental Delivery performed in Scrum-like sprints
- If requirements are vague, might use Iterative Refinement to clarify requirements, followed by Modified Waterfall to build final version (prototype is discarded)
  - Might Integrate & Configure a COTS platform for prototype
- Might develop user interface with iterative refinement, but adopt another process for data store

Phased development

- Decide at the outset to divide a project into multiple phases
  - First phase product is quickly brought into (limited) production
  - Subsequent phases based on experience from first phase
- Advantages
  - Early benefit from initial investment
  - Clarifies requirements for later phases
  - Costs can be spread out (or subsequent phases can be cancelled)

Poll

What methodology was used for the FAA AAS?

Was this an appropriate choice?

PollEv.com/cs5150sp26

Summary

- Different development processes are appropriate for different projects
  - Processes can evolve during a project
  - Processes include common process steps
  - Processes must accommodate revision of prior steps
  - Beware buzzwords
- Purpose of process is to minimize risk. Risk-reduction practices include:
  - Prototyping key components
  - Frequent releases, or decomposition into phases
  - Early and iterative testing with users/customers
  - Promoting visibility

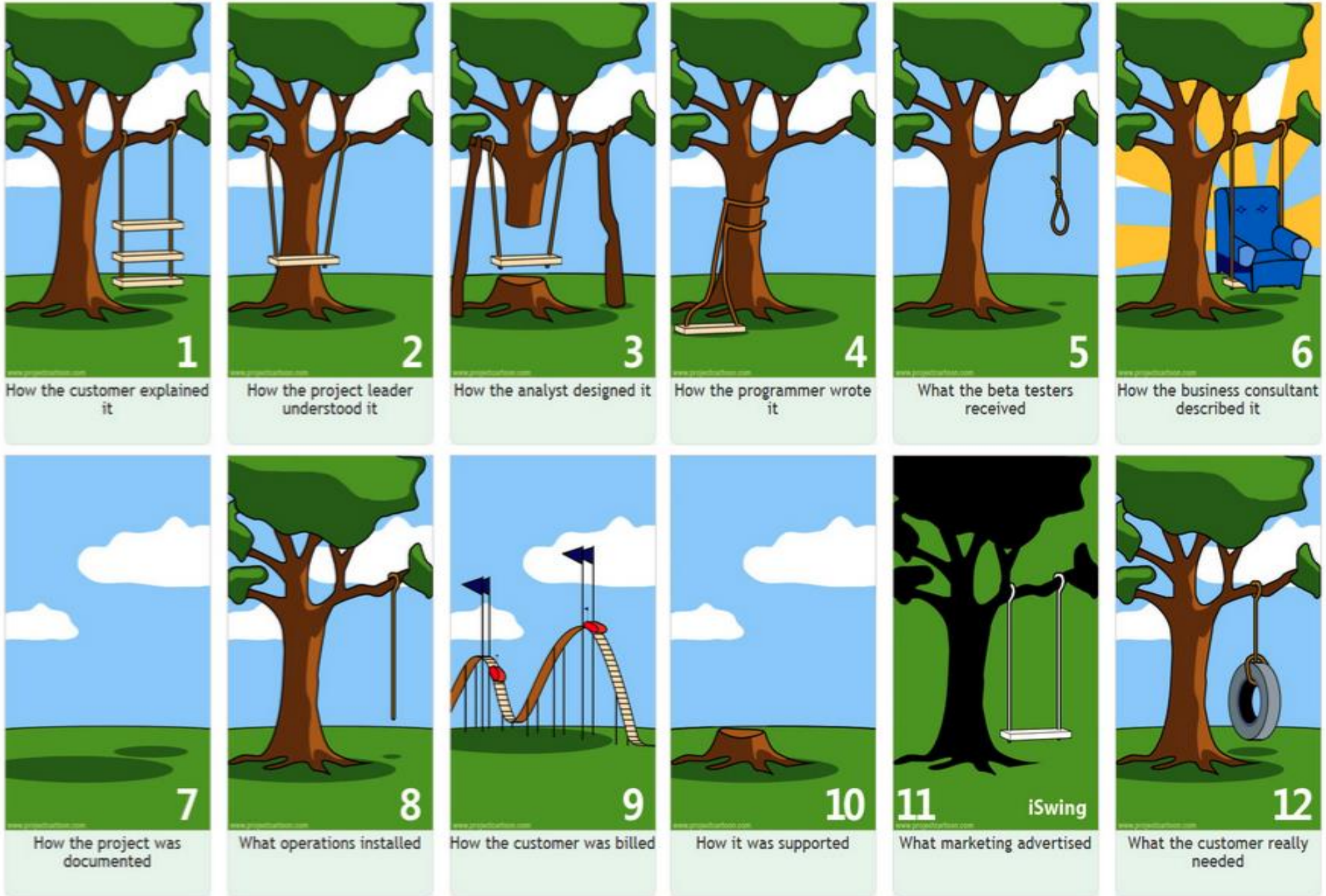
Summary

- Heavyweight: Discourages change; more effort upfront to be confident in design choices
  - Beneficial if system has many inter-related components
  - Example use: Lockheed Martin
- Lightweight: Accommodates requirements uncertainty
  - Iteration can clarify requirements
  - Agility can respond to novel markets
  - Example use: Amazon

Requirements

Lecture goals

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


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

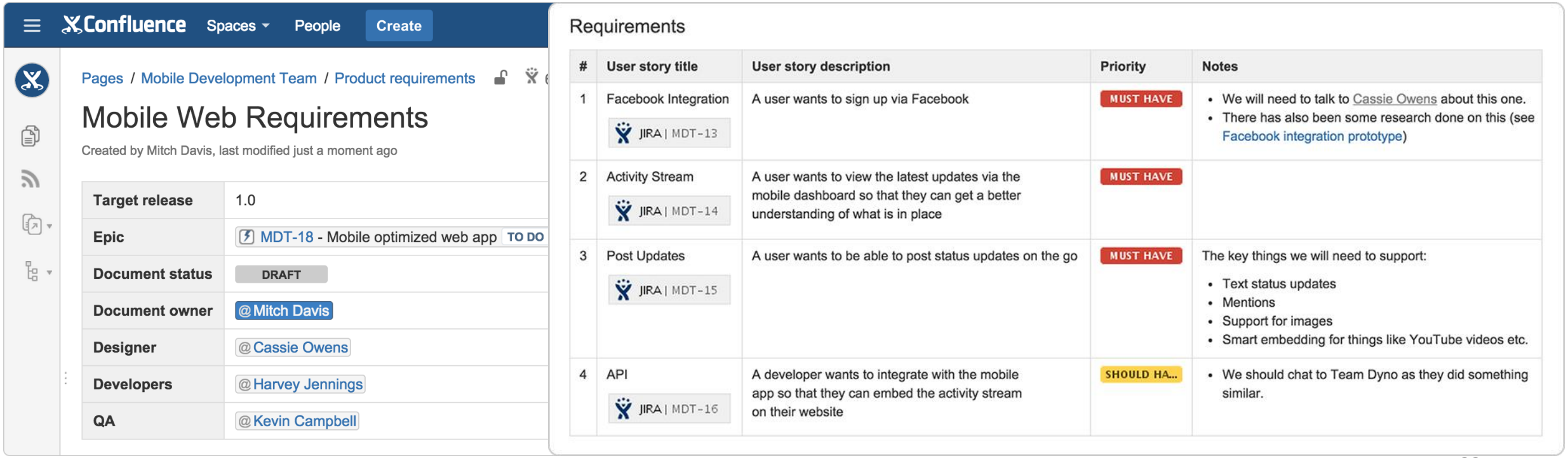
Incomplete requirements	13.10%
Lack of user involvement	12.40%
Lack of resources	10.60%
Unrealistic expectations	9.90%
Lack of executive support	9.30%
Changing requirements & specifications	8.80%
Lack of planning	8.10%
System no longer needed	7.50%

- Failure to understand the requirements led developers to build the wrong system
- 70%: related to requirements and user interactions

The CHAOS Report (1994), The Standish Group

Example

- Also known as Product Requirements Document (PRD)



https://www.atlassian.com/agile/product-management/requirements

Example

- Includes what you are “not doing” as well

Goals

- Our goal is to create a mobile version of the website. Sometimes users click on a link in an e
  notification using their mobile phone and need to be able to access our application from mob
  Chrome or Safari.
- We want to meet feature parity with most functions - except we can skip creating events.

Background and strategic fit

We all know mobile is on the rise. A recent survey to customers showed that 85% of users use the
mobile on a daily basis. Most of our customers also use competitor apps, so this is something we r
to have. We will be able to measure our success through analytics and can use the website today
baseline.

Customer research

- Customer interview - Netflix
- Customer interview - Homeaway
- Customer interview - Bitbucket

User interaction and design

Description	Login screen	Activity stream
Mockup		

Questions

Below is a list of questions to be addressed as a result of this requirements document:

Question	Outcome
What about Google Apps	• We think this is important, but not for version one. • We can look at this at a later stage. • It might be worth someone looking into a shared notification library to do this.
Are we supporting Blackberry?	• Again, not for initial version - but we haven't had much demand for this.
Should we have an offline mode?	• We've talked about the pros and cons. In brief: Seamless experience for customers, they won't notice if there is a connection issue Most of our competitors don't have this Could be expensive to build Should we spike this at a later sprint?

Not Doing

- Google Apps Authentication - out of scope, see above for details
- Blackberry support - we won't look at doing this, if demand picks up we can look at it.
- Native app. We are starting with a mobile web view first and get back to a native app depending on feedback that we get.

Like Be the first to like this requirements

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

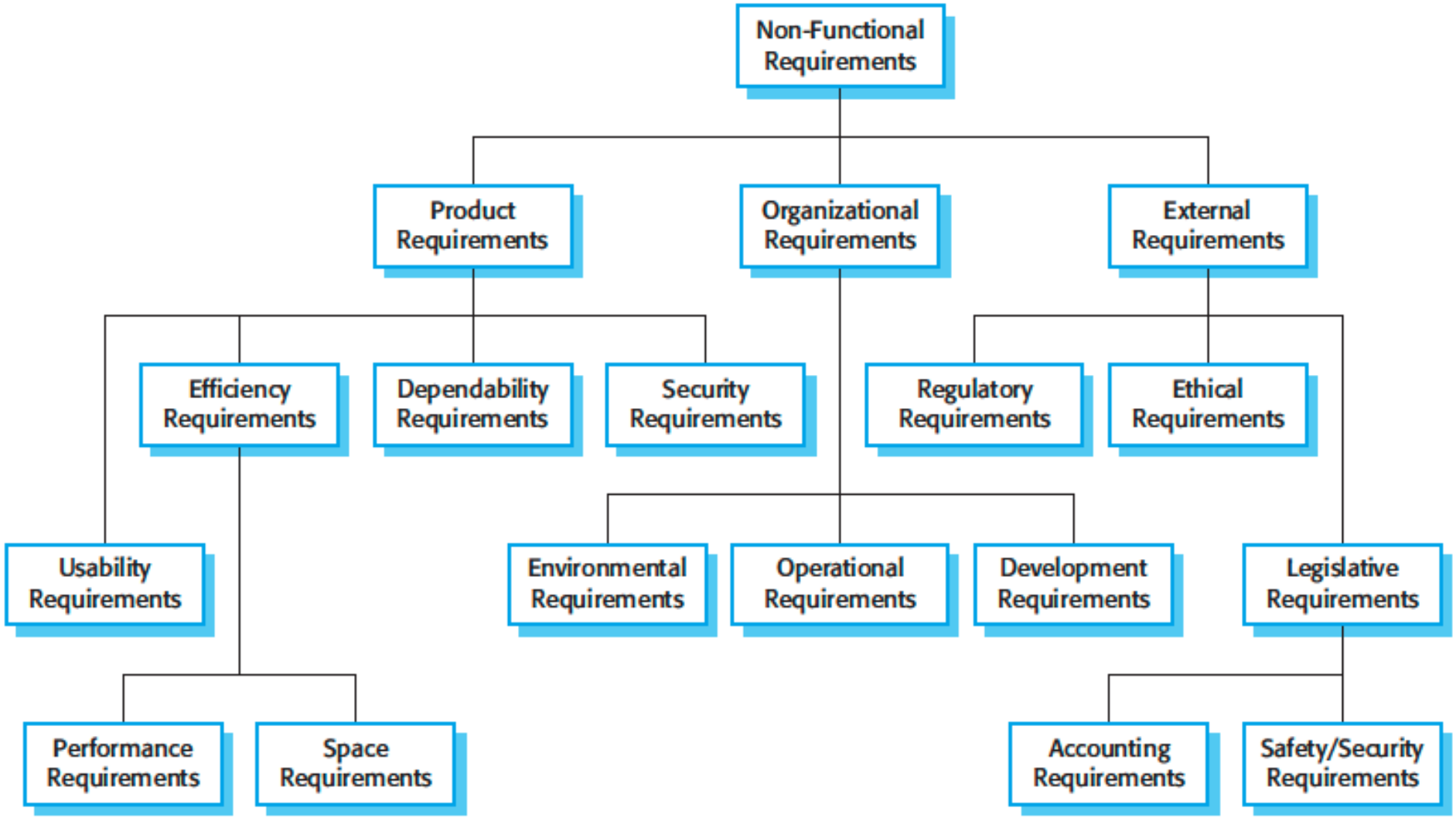
- Start with system-level requirements
- Expand and refine requirements iteratively (e.g., for each sprint)
  - Continual client interaction

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?

- Games for Kids vs Teens
- Operating System/Device constraints
- Works well on trackpad
- Supporting multiple devices/resolutions (high throughput/low latency – refresh rates/latency)
- Online vs offline – resource constraints

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/cs5150sp26

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."

Poll: What kind of requirement is this?

- “We should migrate all our cloud-based backend services from Azure to AWS”

- A: Functional
- B: Efficiency Requirements
- C: Ethical Requirements
- D: Development Requirements

PollEv.com/cs5150sp26

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				

Activity: 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

Brainstorm: Viewpoints for university admissions system

Applicants
Admissions office
Financial aid office
Special (athletic, development) offices
Academic departments
Tech support
operations and maintenance