Lecture 2: Projects & Processes

CS 5150, Spring 2026

Administrative Reminders

- Project Team Matching Survey
- Project Pitch
- Team-forming threads
- See internal project descriptions

- Firehose upfront
  - Need to cover all the basics so you can write your project plan
  - Concepts are high-level, abstract; try to correlate them with a concrete example (like FAA AAS)

Project

- How do I pick a project?
  - Consider this as an opportunity to learn something new (e.g., new language)
  - Do not go into a project where you are not familiar with anything!

- How do I pick a team/teammates?
  - Consider working style preferences, program,
  - Identify complementary skill-set (front-end/backend, source/target language)

- See team matching survey responses: https://docs.google.com/spreadsheets/d/1ySghKRdIm0UGQYlwlOJYWPp53CiSUAP4zaVRWrRDlqI/edit?usp=sharing

Project Options

- Internal projects + internal feature (Client: course staff)
- Internal projects + different feature (Client: course staff)
- Non-internal project + feature (Client: course staff)

- Pre-approved external project + feature (Client: fixed external client)
  - See external-clients.txt (currently 3 projects)
- Other external project/client (E.g., if you know the project owner)
- All cases require approval from instructor/course staff!

Variety

Software is required to serve many different purposes …

- Control systems (vehicles, industrial processes)
- Embedded (appliances, medical devices, remote monitoring)
- Operating systems & drivers
- Developer tools (IDEs, frameworks, compilers)
- Data processing (billing, benefits)
- Information systems (databases, digital libraries, search)
- Commerce (shopping, advertising)
- Science (weather forecasting, data analysis)
- Engineering (CAD/CAM, FEA, EDA)
- Multimedia & entertainment (video conferencing, games, VR/AR)
- Creativity (3D modeling, photography)
- Productivity (spreadsheets, desktop publishing)

Variety (cont.)

… in many different settings …

- Embedded firmware
- RTOS
- PC
- Smartphone
- Web browser
- Supercomputer
- Virtualized servers
- Cloud

… for many different people.

- Yourself
- Consumers
- Professionals
- B2B
- Employer/colleagues
- Government agencies
- Prime contractors
- General public

… requires versatility

Consequently, there is no “best” way to create software in all cases

- No best operating system
- No best programming language
- No best framework or architecture
- No best development environment/tools
- No best **methodology/process**

A software engineer must know a wide variety of methods & tools and select appropriate ones for the project at hand

Project stakeholders

- First step in any project:
  identify the stakeholders
  - Who sets requirements?
  - Who decides priorities?
  - Who will use your software?
  - Who is affected by your software?
  - Who writes the check?
  - Who takes the fall?
- Stakeholder interests are not always aligned

Stakeholders: Developers

- You are a stakeholder
  - You have to work with the code
  - You have to support the system
  - Your reputation is on the line
- You are also an (expensive) resource
  - **Biggest cost of software is salaries of development team**
- You have responsibilities
  - Competence
  - Confidentiality
  - Legal compliance (e.g., FERPA)
  - Acceptable use & misuse

Stakeholders: Client

- Provides resources in exchange for having the software developed
- Bears risk in event of project failure

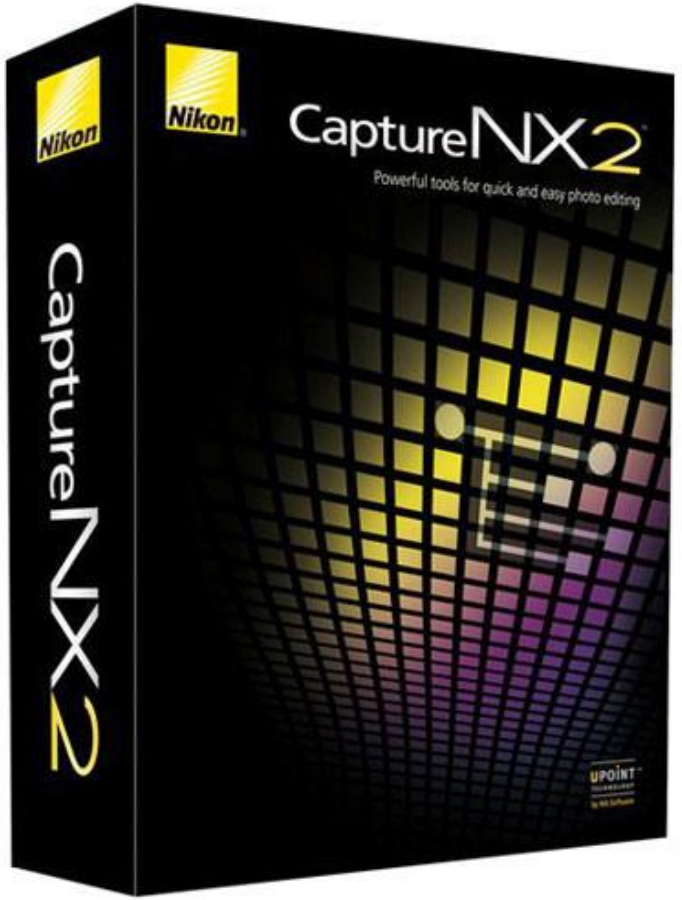
- Client sets requirements
  - Though developers must elicit them
- Client sets priorities

- **Client satisfaction is primary measure of project success**

Example: business-to-business

Nikon contracts with Nik Software to co-develop “Nikon Capture NX”, a digital photo editor sold to users of Nikon cameras

- Developer: Nik Software
- Client: Nikon (specifically, a product manager in their imaging business division)



Poll

Who is the client for general-purpose software products?

PollEv.com/cs5150sp26

Stakeholders: Customer, User, Society

- Customer: buys the software or selects it for use by an organization
- User: Actually uses (interfaces with) the software
- Society: may be affected by the software
  - Often not represented when stakeholders are consulted
  - Advisable to appoint an *advocate* for their interests
    - Automated processes tend to become invisible
  - Risks to society should be identified and acknowledged
  - Example: Cornell's Outlook Exchange Service

Activity: Stakeholders

1. Read FAA AAS case: https://sebokwiki.org/wiki/Federal_Aviation_Administration_(FAA)_Advanced_Automation_System_(AAS)
2. Turn to your neighbor
3. Identify the stakeholders (developer, client, customer, user) for:
    1. canvas.cornell.edu
    2. FAA's Advanced Automation System
4. Select a reporter to share results

(3 minutes)

Risk

- All projects require tradeoffs between **function**, **cost**, and **time**
- Many projects encounter difficulties:
  - Does not work as expected (function)
  - Over budget (cost/resources)
  - Behind schedule (time)
- Who should set priorities when deciding tradeoffs?
  - The client bears the cost of the project
  - The client bears the risks of project failure
  - The client should be given the information necessary to make an informed decision based on *their* priorities

Consequences

- Failed projects have serious consequences
  - Can bankrupt companies
  - Managers can lose their jobs
  - Users and society may be harmed

- Example: Apple Maps 2012; Maps chief fired

Apple Maps service loses train stations, shrinks tower and creates new airport

Significant glitches reported in service that replaces Google Maps on Apple's iOS6 for iPhones and iPads

https://www.xda-developers.com/apple-maps-launched-11-years-ago
https://www.theguardian.com/technology/2012/sep/20/apple-maps-ios6-station-tower

Minimizing risk – communication

- As much as half of delivered software is never used
  - Developers build the “wrong software” – doesn’t meet client’s needs
- Developer must work to understand client, customer, and user expectations
- Developer may add technical insights, but **client satisfaction is the primary measure of success**

Minimize risk with communication

- Feasibility study
- Requirements and design (separated)
- Milestones & releases
- User & acceptance testing
- Handover

Minimizing risk – visibility

- Those responsible for the project (client, managers) must know what is happening
- But most developers …
  - Have trouble evaluating progress
  - Tend to be overly optimistic
  - Consider logging/reporting to be unnecessary overhead
- Large projects are worse
  - Dilution at every level of hierarchy
- In CS 5150, you will provide visibility via regular progress reports
- Working software provides good visibility
  - Promoted by Agile methods
  - But be upfront about limitations

Improving visibility – short dev cycles

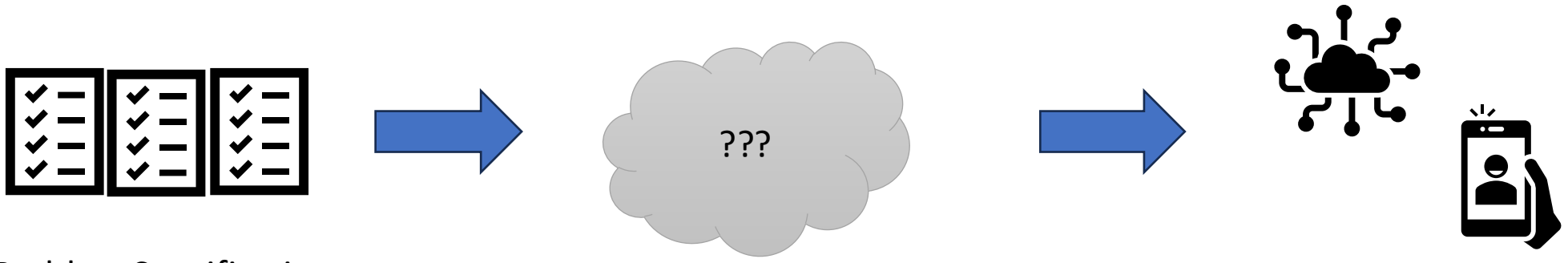
- Risk accumulates with time since last check-in

- Deliver working software frequently (weeks rather than months, or even continuously)
  - Clients, customers, & users can evaluate work
  - Opportunity to adapt to new circumstances
  - Promoted by Agile methods

Minimizing risk – management

- Project management
  - Track progress against schedule
  - Prioritize tasks
- Personnel management
  - Allocate the right number of developers with the right skills at the right time
  - Ensure that developers have a productive work environment
- Compliance advising
  - Understand legal, regulatory, economic environment
- Development processes
  - Enforce best practices to minimize risk without excessive overhead
  - Improve visibility
  - Facilitate team productivity
  - Ensure quality

The software development challenge

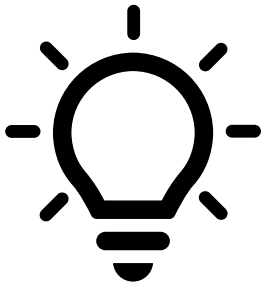


Problem Specification

Solution (product, app, …)

One solution: Code and fix

Specification
(maybe)



Deliver?
(maybe)

One solution: Code and fix

- Pros:
  - Little to no overhead – just dive in and develop, see progress quickly
  - Applicable sometimes for small projects, short-lived prototypes, and/or small teams

- Cons:

One solution: Code and fix

- Pros:
  - Little to no overhead – just dive in and develop, see progress quickly
  - Applicable sometimes for small projects, short-lived prototypes, and/or small teams

- Cons:
  - No way to assess **progress**, **quality,** or **risks**
  - Challenging to manage multiple developers – how synchronize your work
  - Harder to **accommodate** changes w/o major design overhaul
  - Unclear **delivery** of features (scope), timing, and support

Is a more structured process necessary?

It establishes an order – provides a model – of software project events

- Forces us to *think* of **"big picture"** and follow steps so that we reach it without glaring deficiencies
- W/o it we may make decisions that are individually on target but collectively misdirected
- Allows us to **organize** and coordinate work as as team
- Allows us to **track** progress and **minimize** risks, and adjust as necessary

Overview of development process steps

- Feasibility
  - Define scope
  - Catalog benefits, risks
  - Evaluate technical feasibility
  - Select development process
  - Estimate cost, schedule, resource availability
  - Decide: go/no-go

- Requirements
  - Define function of system *from client's viewpoint*
  - Establish constraints ("non-functional requirements")
  - Elicit from consultation with client, customer, users
    - Self-contained study or incremental

- Biggest cause of failed projects

Overview of development process steps (cont.)

- System & interface design
  - Select an architecture that supports requirements
  - User interfaces must be iteratively evaluated with users
  - Architectural integrity is key to maintainable systems
- Program development
  - May start with documenting program design (class & function definitions)
  - Coding!
    - What you already know how to do
  - May incorporate testing

Overview of development process steps (cont.)

- Acceptance & release
  - Product is verified against requirements *by the client*
    - Ideally with selected customers & users
  - Complete system (with documentation) delivered to client
    - Deployed in production, marketed to customers
- Operation & maintenance
  - System is kept running smoothly
  - Bugs discovered and fixed in production
  - New features proposed and integrated (requirements change)
  - May eventually be phased out

Key question: How to combine the stages and in what order?

Activity: FAA AAS Discuss

Which steps were handled poorly for the FAA's Advanced Automation System?

(3 minutes)

Feasibility & planning

Requirements

System & interface design

Program development

Acceptance and release

Operations and maintenance

Software Methodologies

- Can organize sets of process decisions by how they address the common process steps
  - Formal vs. informal
    - Do steps have pre-defined outputs?
  - Duration and ordering
- Heavyweight
  - Fully complete (and document) each step before moving on
  - Avoid revisions to work done in previous steps
- Lightweight
  - Schedule work in “time boxes” that include multiple process steps
  - Avoid formal documentation to more easily accommodate changes

Heavyweight vs lightweight methodologies

Heavyweight

- Processes and tools
- Specifications
- Following a plan
- Client negotiation

Lightweight

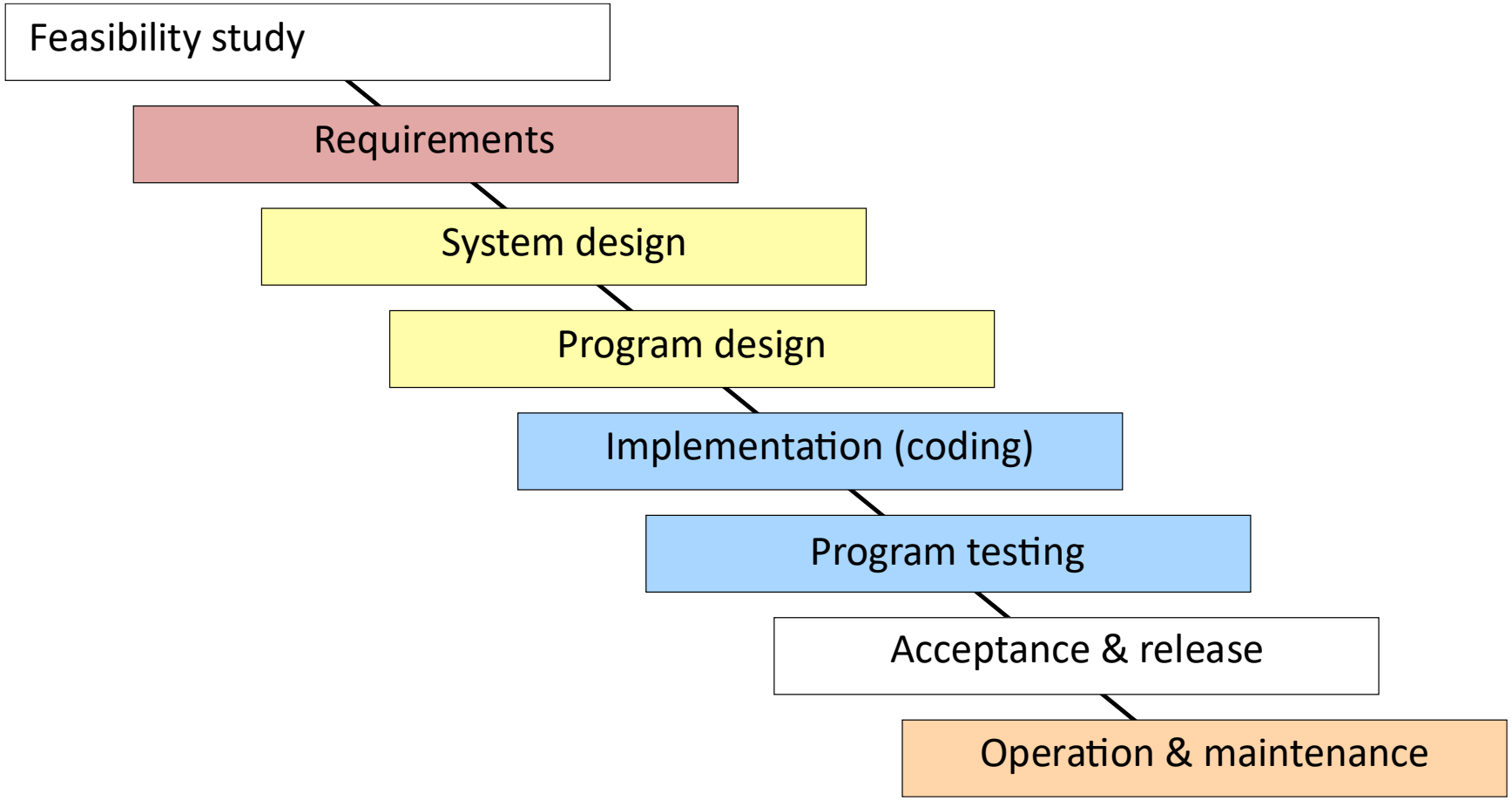
- Individuals and interactions
- Working software
- Responding to change
- Client collaboration

Based on the Manifesto for Agile Software Development
http://agilemanifesto.org/

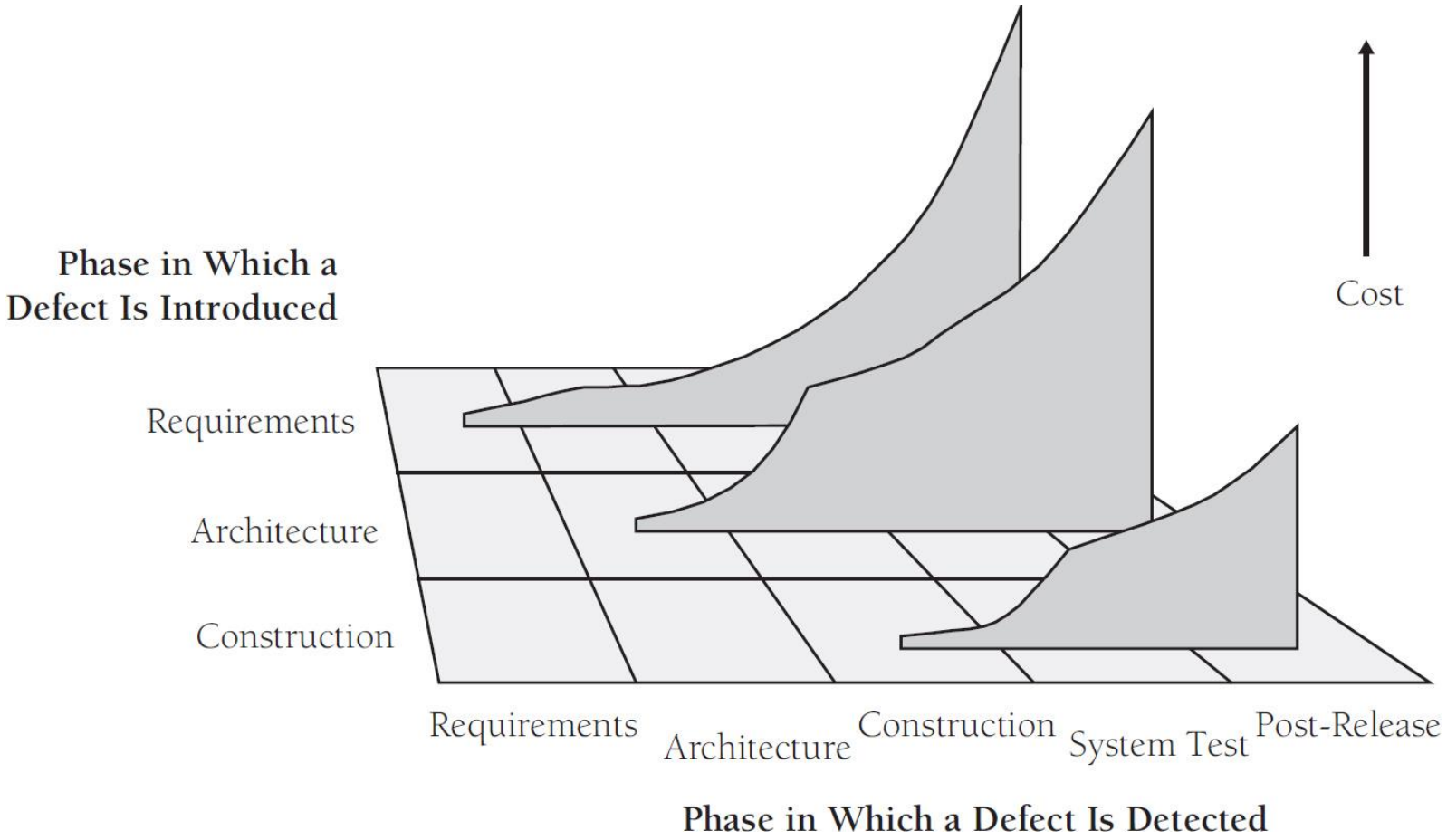
Waterfall model: Origins

- Based on traditional engineering project management
  - Long lead time for supplies; must commit to large orders
  - Extremely expensive to change hardware once built, BoM once ordered
  - Extremely expensive to pause manufacturing
- At this time in software history,
  - Requirements well understood (automating manual processes)
  - Little variety in system design
  - Coding was very tedious (no modern languages/tools) – benefits from detailed program design
- Good match for a heavyweight process

The waterfall model



Cost of defects



Code Complete, 2nd Edition. McConnell 2004

Shift left

- QA is difficult without a working system
  - But working systems aren't available until the end of a waterfall process
- Process decisions can effectively shift QA left without requiring formal deliverables after each step

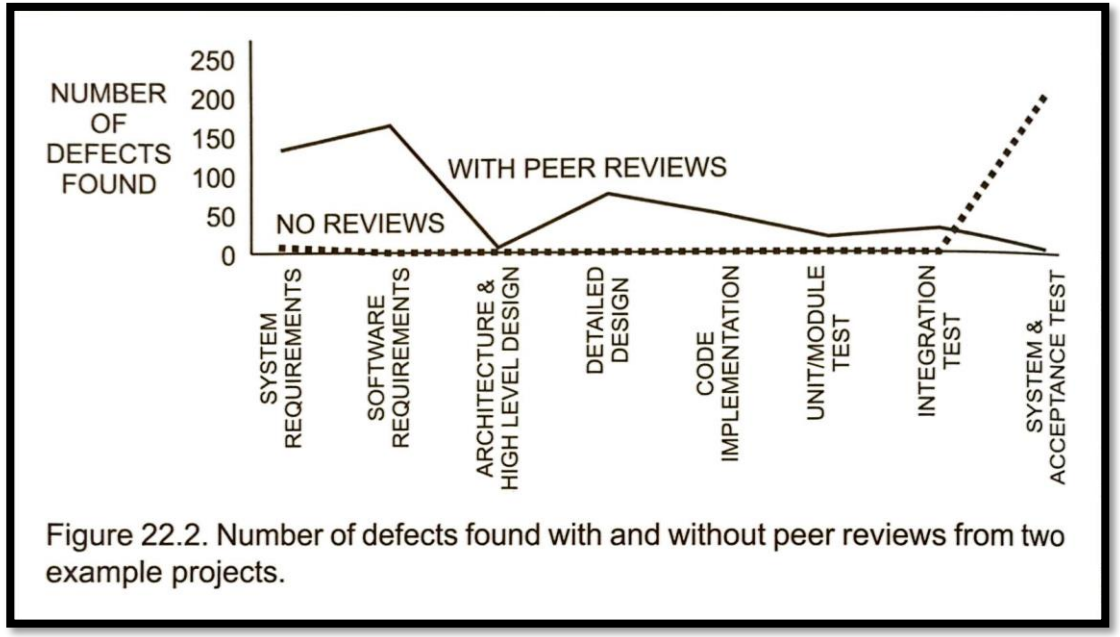


Figure 22.2. Number of defects found with and without peer reviews from two example projects.

Better Embedded System Software. Koopman 2010

The waterfall model

Advantages

- Separation of tasks
  - Aids personnel management
- Process visibility
- Quality control at each step
- Cost monitoring at each step

Disadvantages

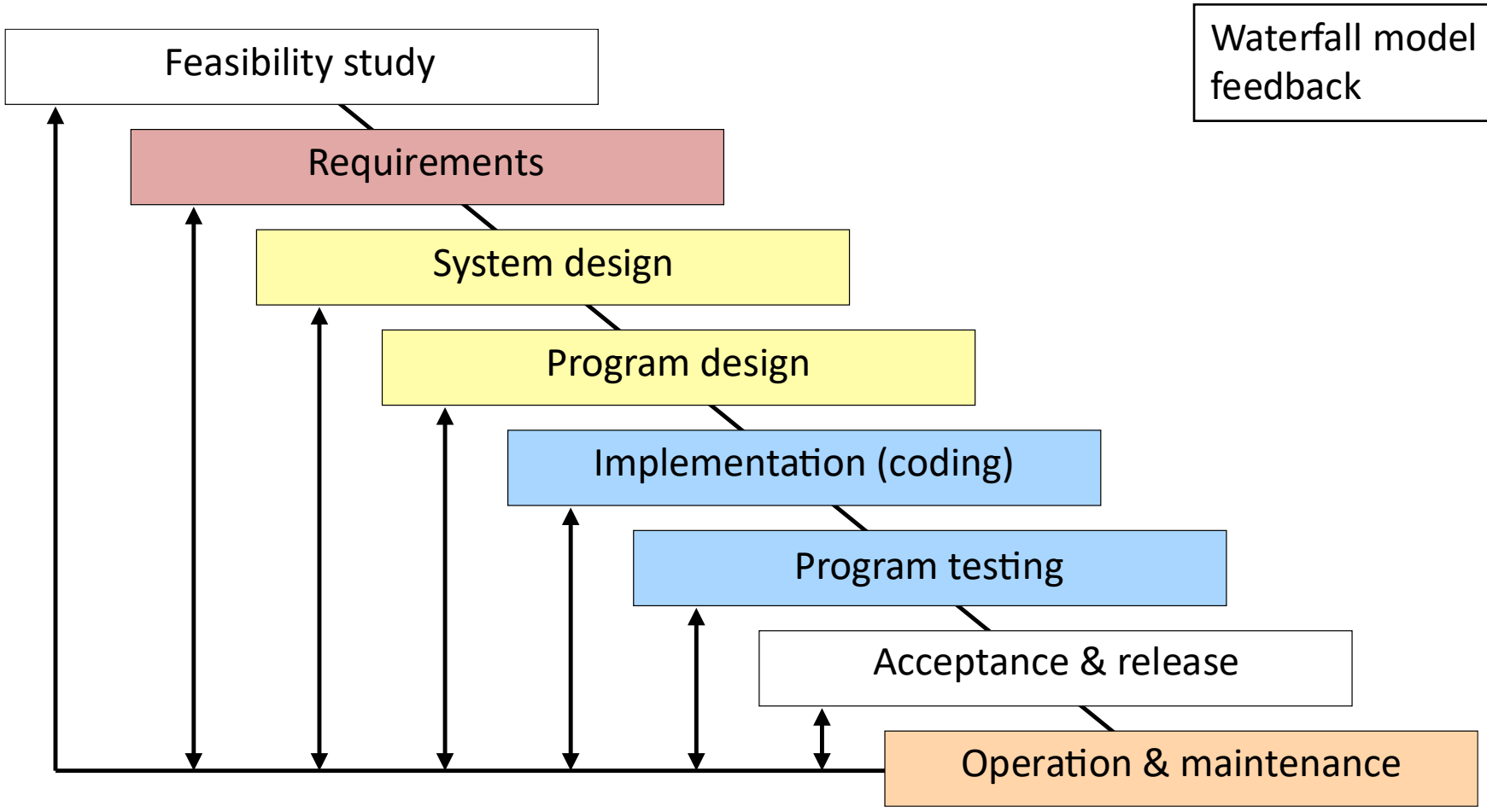
- In practice, later stages improve understanding of earlier stages, necessitating revision
- Not flexible enough to react to changing conditions

Question: What is missing?

Iteration is required

- Feasibility study needs preliminary requirements and tentative design
- Implementation often reveals gaps in requirements
- User interfaces hard to analyze without actually using them
- Requirements, technology may change during development
  - E.g. updated market analysis

Modified waterfall model



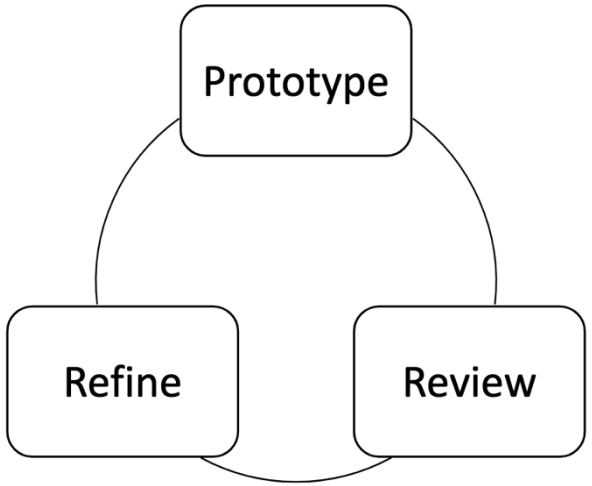
Waterfall model with feedback

Modified waterfall model

- A fine choice when requirements are well-understood and system design is fixed
  - Automating manual data processing systems (e.g. utility billing)
  - New version of system whose functionality derives from earlier product (e.g. embedded controller)
  - Self-contained components/services with a pre-defined interface
- Widely recommended for safety-critical or highly regulated systems
  - Requirements must be thoroughly analyzed and documented
- *Maybe* suitable for CS 5150 projects
  - But plan for iteration around user interfaces

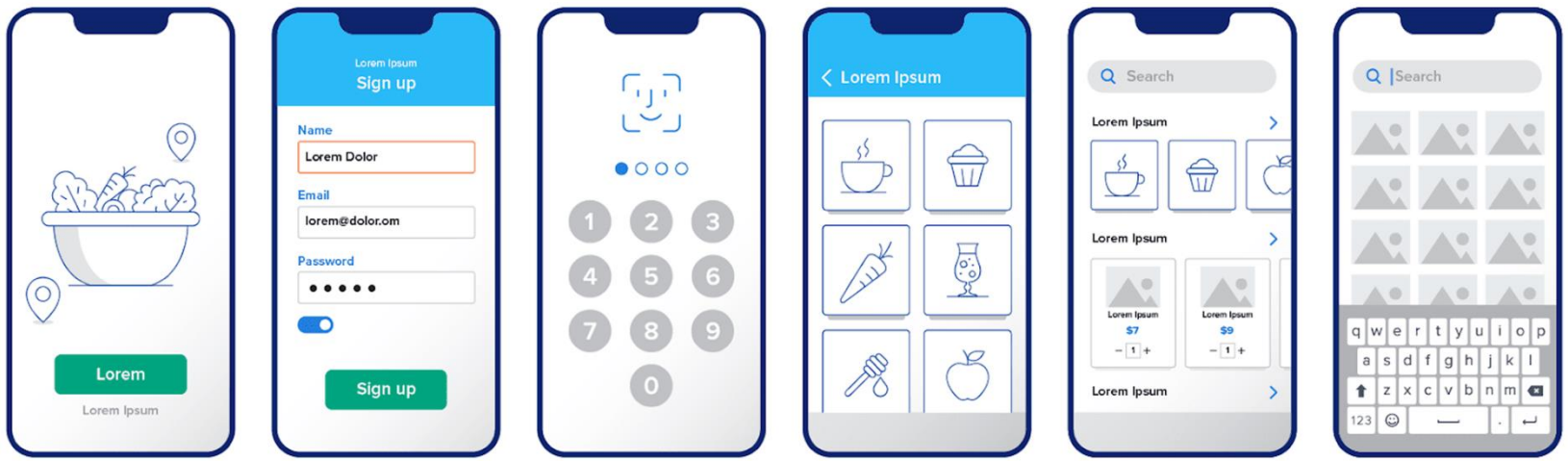
Iterative refinement: Prototyping

- 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?

UI Prototyping is popular



Iterative refinement: Prototyping

- Pros:
  - Client involvement and early feedback
  - Improves requirements and specifications
  - Reduces risk of developing the “wrong” product
- Cons:
  - Time/cost for developing may be high
  - Hard to commit what will be delivered and when
  - May end up evolving a poor choice (limit thinking holistically)