Lecture 2:
Projects & Processes

CS 5150, Spring 2025

Admin Stuff

- Project Team Matching Survey
- Team-forming threads
- See internal project descriptions
- Canvas tour

- 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)

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

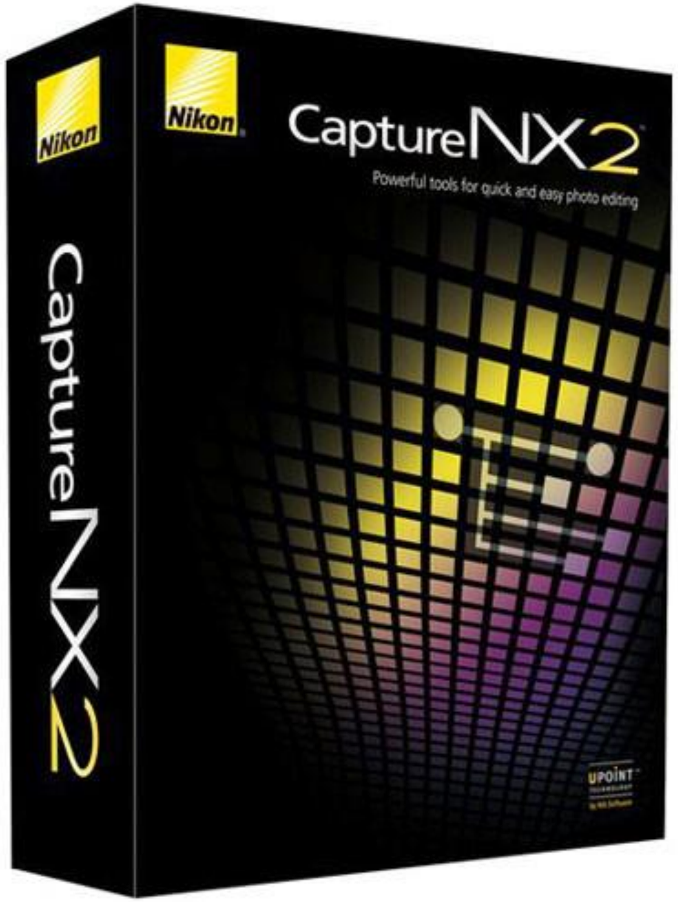
- Provides resources in exchange for having the software developed
- Bears risk in event of project failure

- 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/cs5150sp25

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

Activity: Stakeholders

1. Turn to your neighbor
2. Identify the stakeholders (developer, client, customer, user) for:
    1. canvas.cornell.edu
    2. FAA’s Advanced Automation System

3. Select a reporter to share results

(3 minutes)

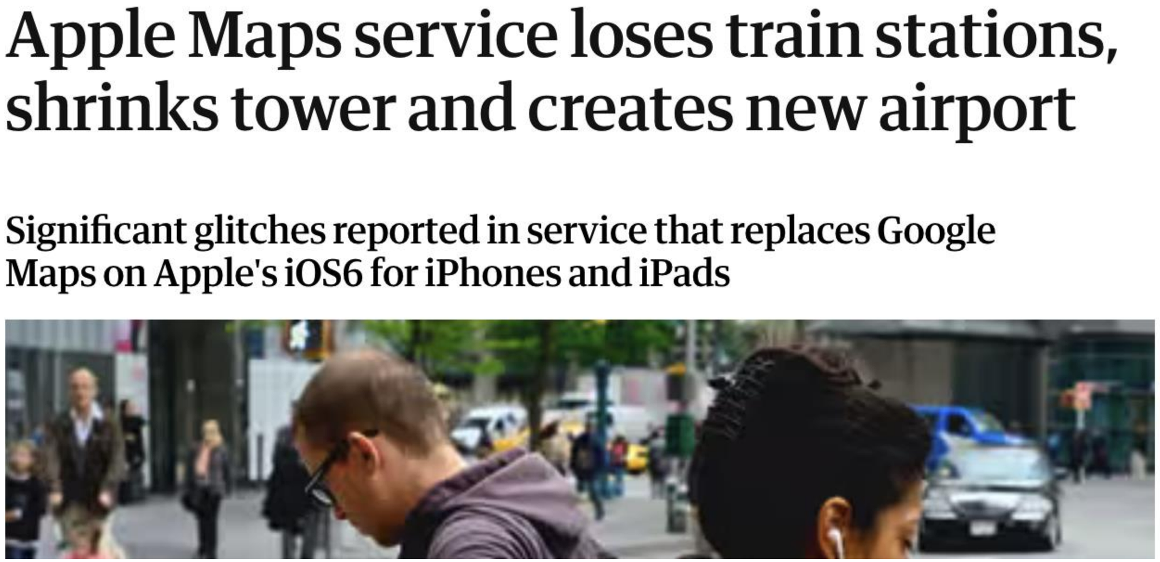
Risk

- All projects require tradeoffs between **function**, **cost**, and **time**
- Many projects encounter difficulties:
  - Does not work as expected (function)
  - Over budget (cost)
  - 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

Development processes

- Example process decisions:
  - How requirements are tracked
  - How tasks, issues are tracked & prioritized
  - How software versions are controlled
  - Code review mandates
  - Test coverage mandates
  - Amount, timing of documentation
  - Frequency, style of meetings
  - What metrics are collected
- Tradeoff between risk reduction and overhead
  - Effectiveness, cost depend on tool support, developer skill, culture
  - Initial risk depends on project size
  - Risk tolerance depends on application
- Must adapt process to each project
- Aim to improve processes throughout project

Process steps

- Project specifics are different, but they need to address similar issues
- Process decisions should be adopted to address common process **steps**

- Note: testing & documentation occur in many steps

- Feasibility & planning
- Requirements
- System & interface design
- Program development
  - Includes program design
- Acceptance and release
- Operations and maintenance

Repeated

Overview of 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 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 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

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	**Lightweight**
• Processes and tools	• Individuals and interactions
• Specifications	• Working software
• Following a plan	• Responding to change
• Client negotiation	• Client collaboration

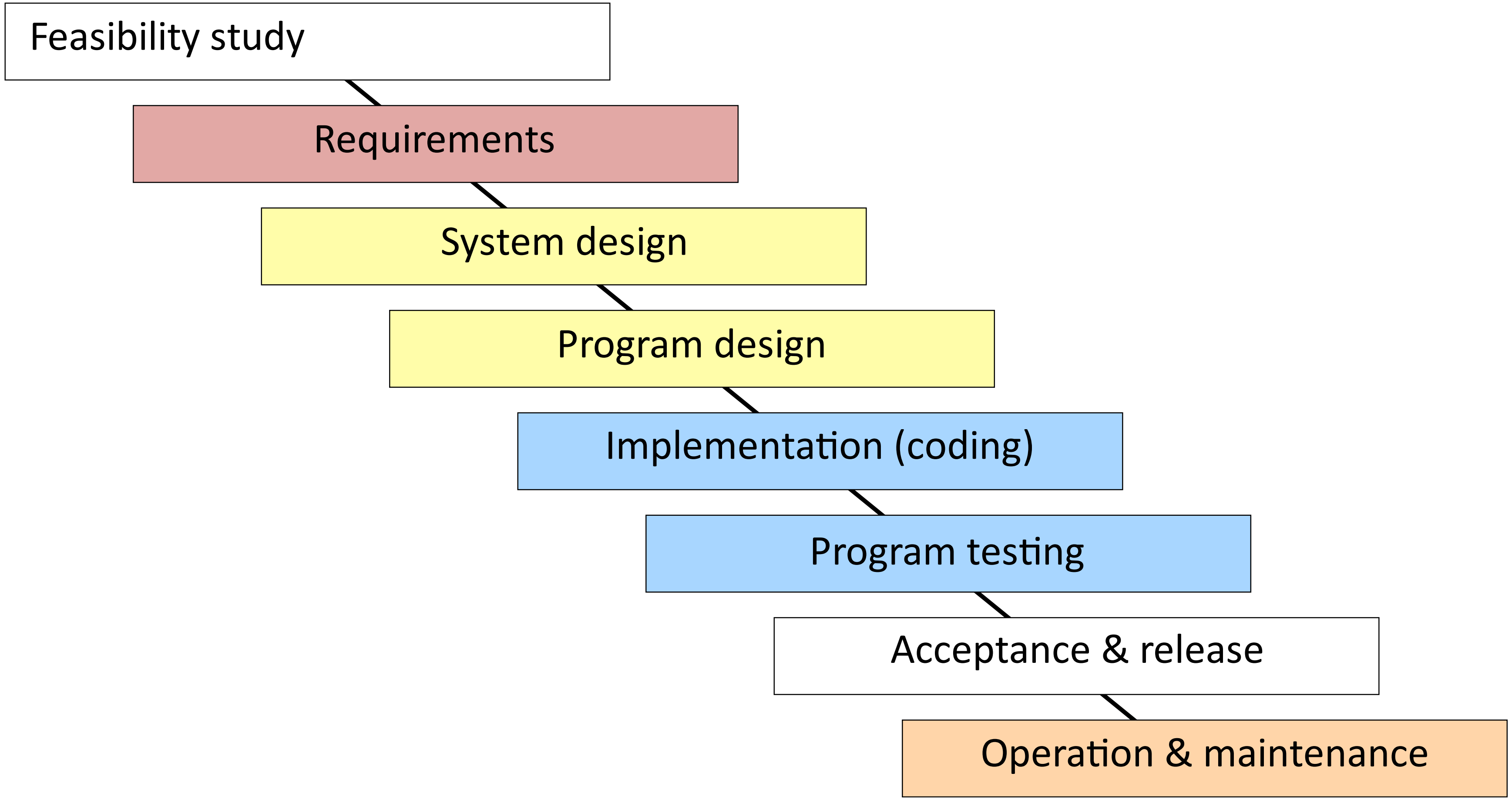
Based on the Manifesto for Agile Software Development
http://agilemanifesto.org/

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



Feasibility study

Requirements

System design

Program design

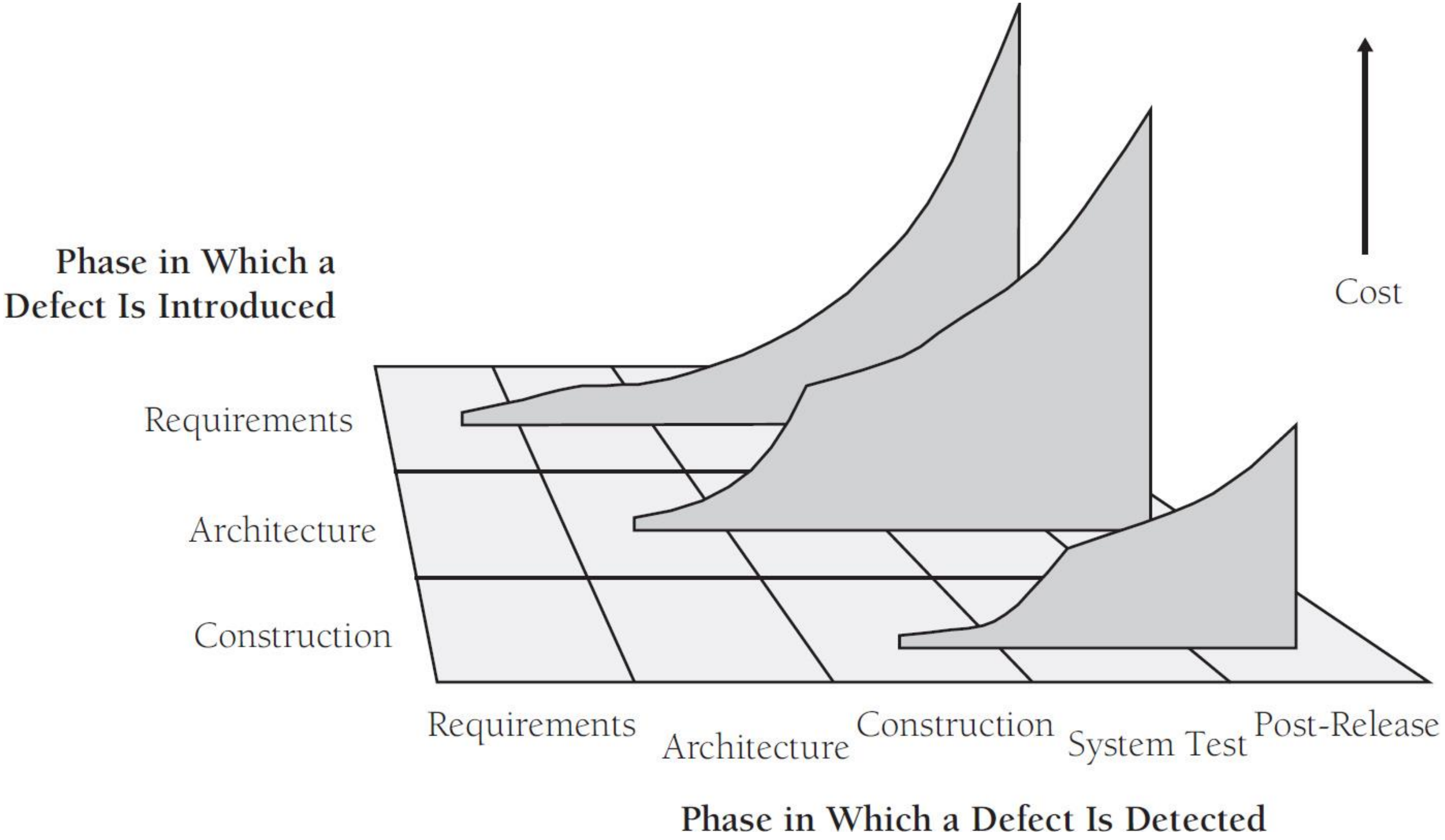
Implementation (coding)

Program testing

Acceptance & release

Operation & maintenance

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

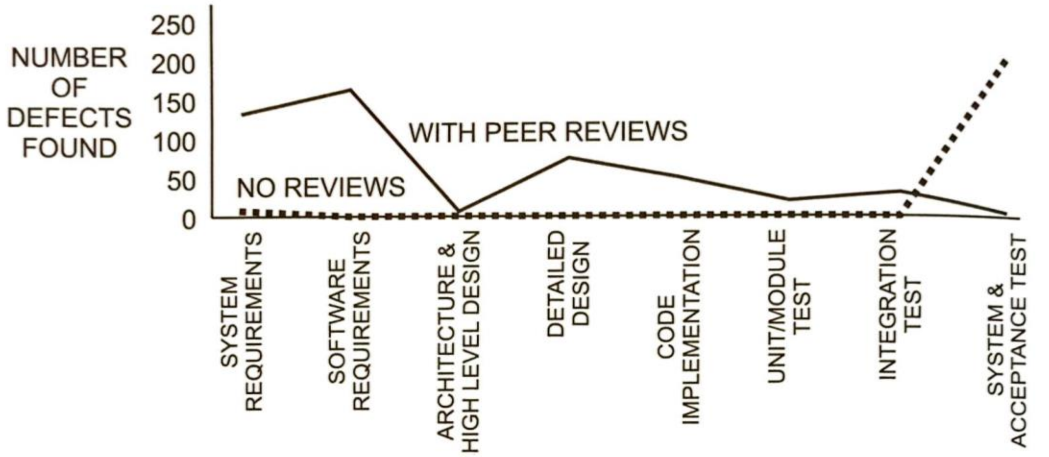


Figure 22.2. Number of defects found with and without peer reviews from two example projects.

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

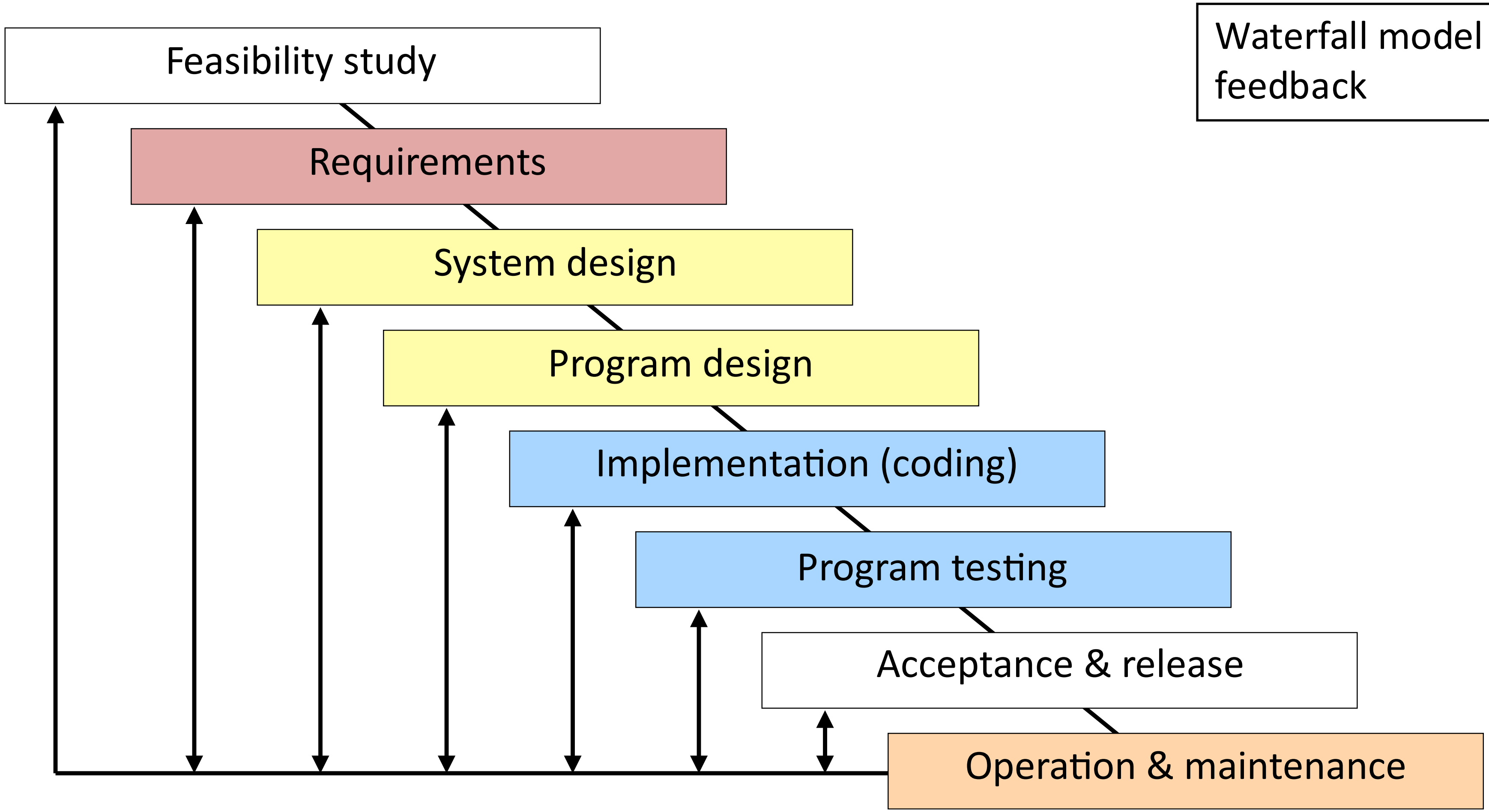
- In practice, later stages improve understanding of earlier stages, necessitating revision
- Not flexible enough to react to changing conditions

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



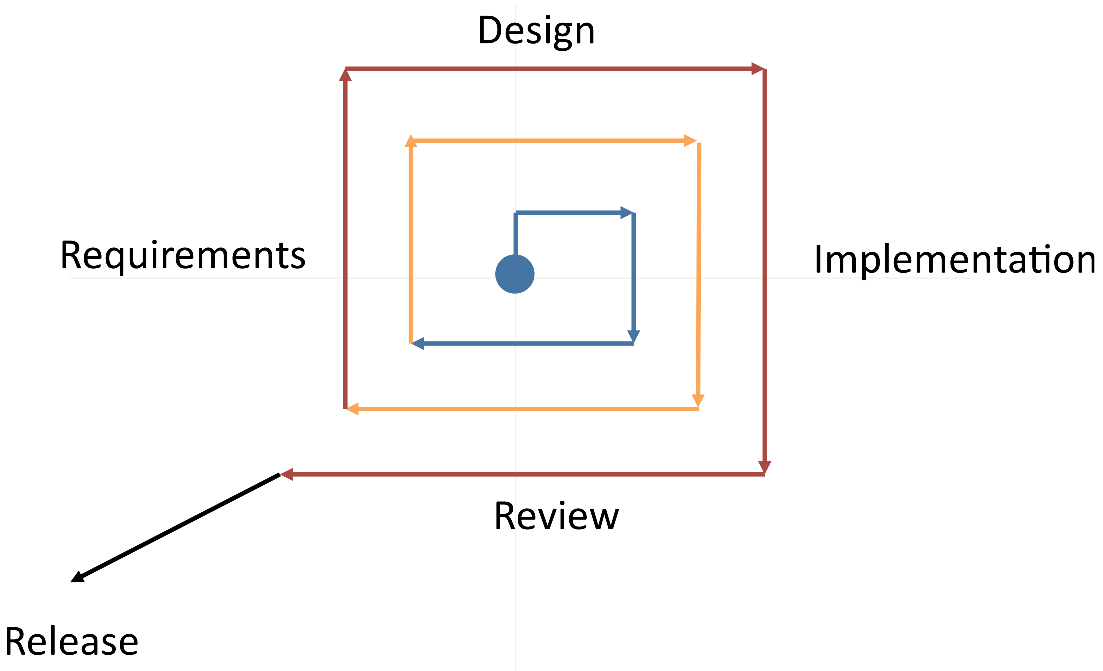
Modified waterfall model

- 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
- Suitable for CS 5150 projects
  - But plan for iteration around user interfaces

Iterative refinement

- 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

Iterative refinement



- Each prototype should be formally evaluated, producing an evaluation report
- Medium-weight process
  - Documentation produced after each review, revised during iterations

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)

Development step decisions

- Overall methodology affects schedule, task assignments, deliverables
    - Management-focused
- Still leaves flexibility in fine-grained development policies

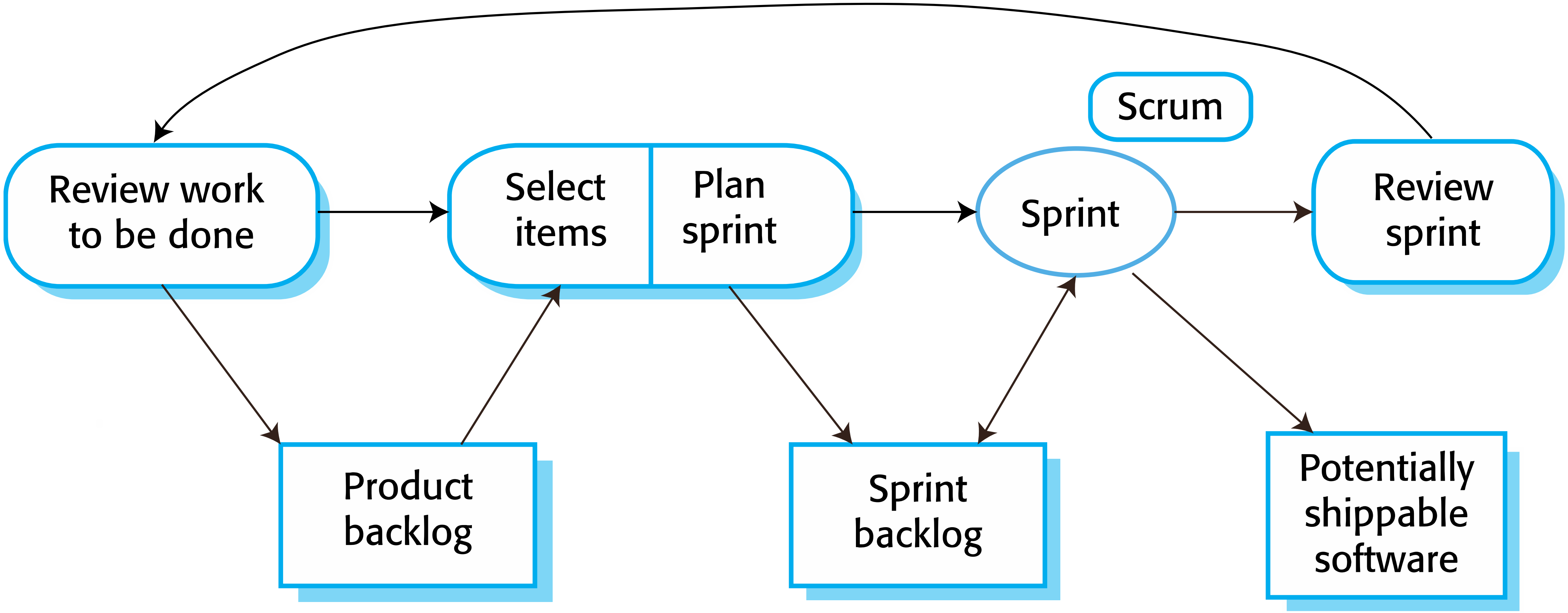
Agile methods and eXtreme Programming

- 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

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



Software Engineering, Tenth Edition. Sommerville 2014

Agile/scrum

Benefits

- Good visibility and communication
- Accommodates change, fuzzy requirements
- Very popular today for small, dynamic projects

Challenges

- Tricky to scale to large projects, bureaucratic organizations
- Works best with highly-skilled, autonomous developers
- Hard to validate requirements for completeness
- Lack of formal docs impedes maintenance, 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

Poll

What methodology was used for the FAA AAS?

Was this an appropriate choice?

PollEv.com/ cs5150sp25

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)

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

Assignment

- Read *Software Engineering at Google*, Chapter 2: How to Work Well on Teams

- Keep forming teams!