CptS 322- Software Engineering Principles I

Software Lifecycle

Instructor: Sakire Arslan Ay

Fall 2021

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

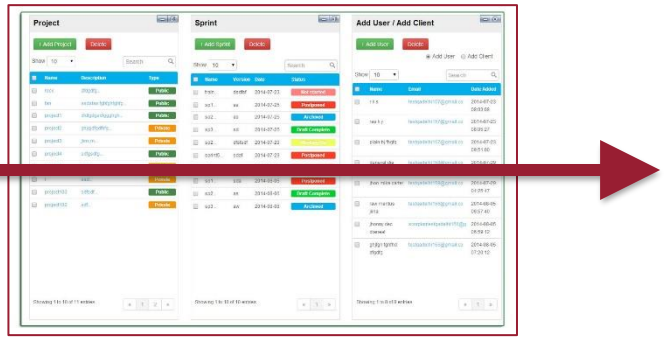
Outline

- Software phases
- What is a software development lifecycle?
- Why do we need a lifecycle process?
- Five basic lifecycle models and their tradeoffs
- Evaluating models

Questions?

- What is Software Engineering?
- How is it different than programming?
- Why do we need it?
- Why is it too hard to build **good** software?

Software Development



Software Process:

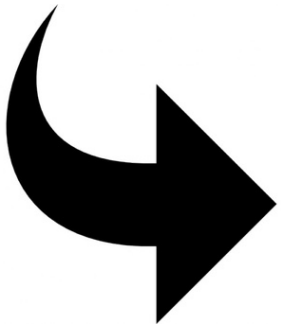
- Systematic
- Formal (or semi-formal)

Discipline of Software Engineering

- methodologies

- techniques

- tools



High quality software that:

- works
- fits the time and budget

Software Development Effort

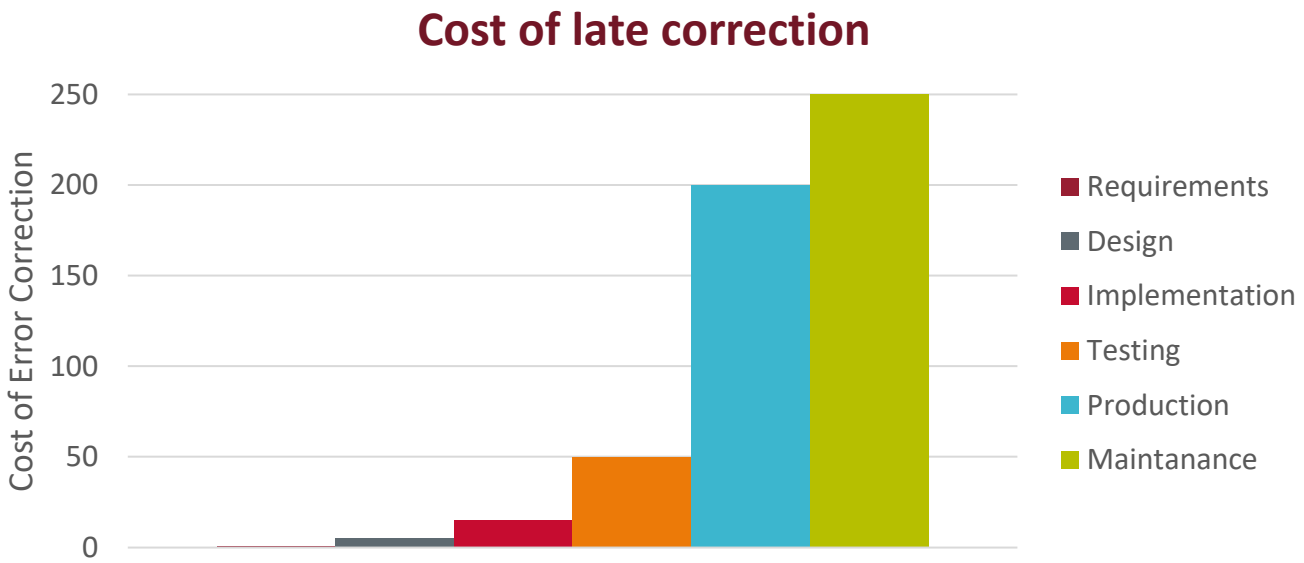
Lines of code:	Task:	
10^2	– Class assignment	Programming Effort
10^3	– Small project	
10^4	– Term project	
10^5	– Word processor	Software Engineering Effort
10^6	– Operating system	
10^7	– Distributed system	
…….	…….	

Software Phases

- Each phase requires different tools, knowledge, skill-set.
- How are these related? What is a good order?

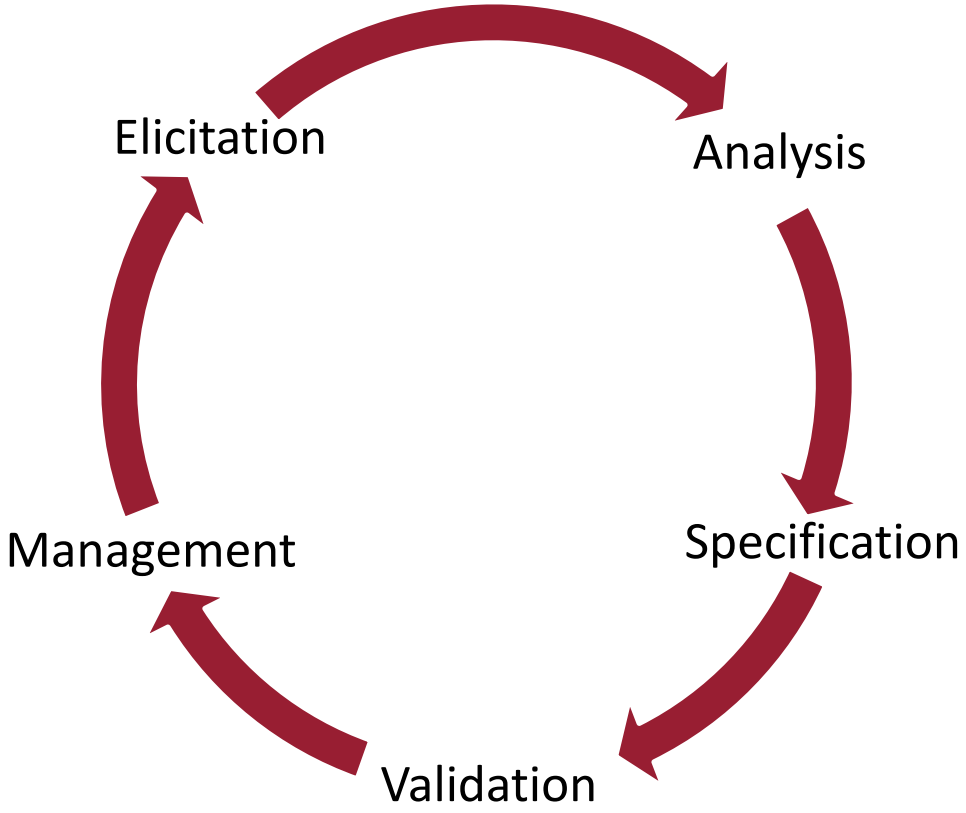
Requirements Engineering

- **Requirements Engineering** (RE) is the process of establishing the needs of stakeholders that are to be solved by software
- Why is this phase important?

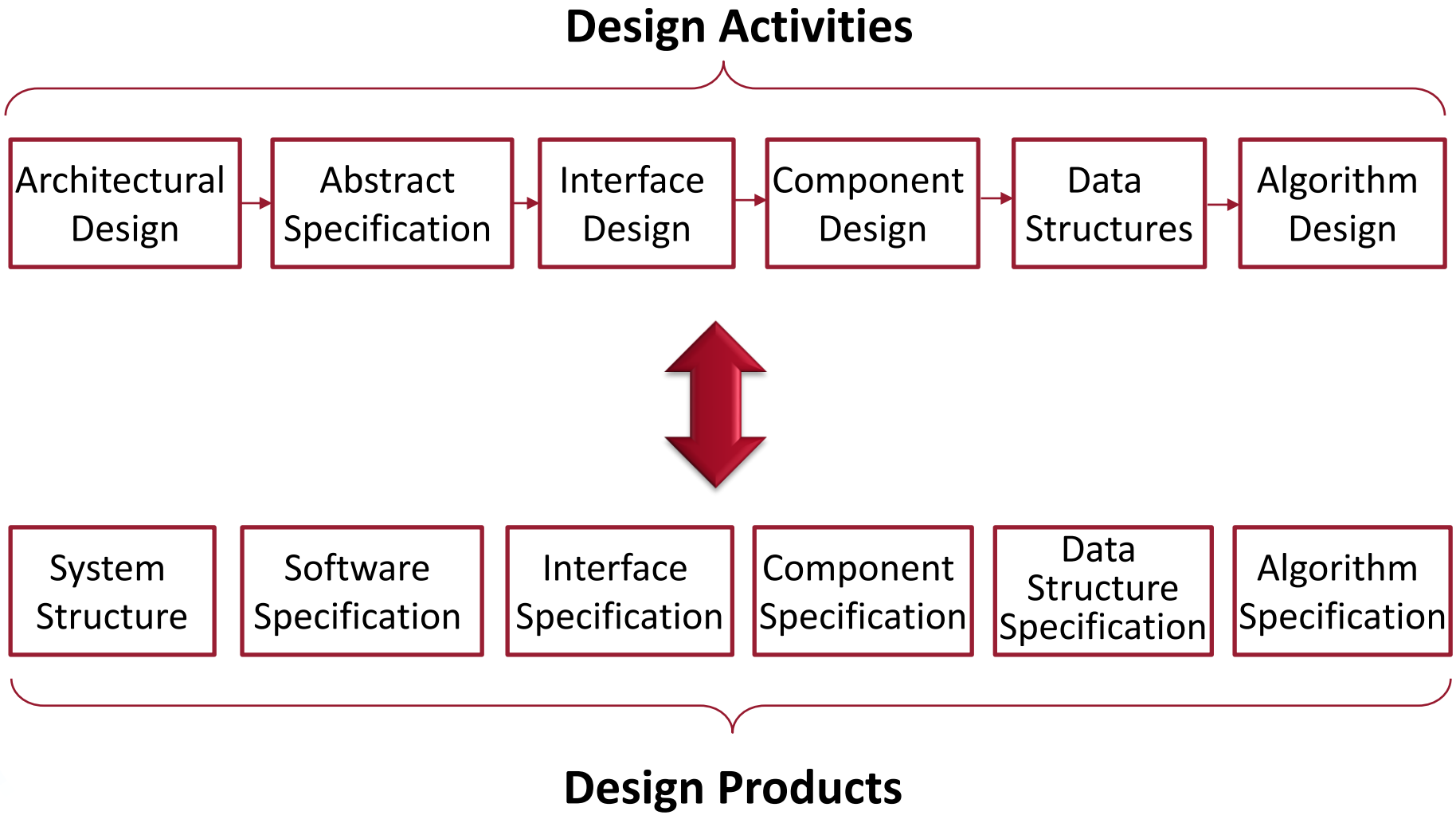


Requirements Engineering

- How do we collect requirements?
  1. Elicitation
  2. Analysis
  3. Specifications
  4. Validation
  5. Management



Design


Design Activities
Architectural Design
Abstract Specification
Interface Design
Component Design
Data Structures
Algorithm Design
System Structure
Software Specification
Interface Specification
Component Specification
Data Structure Specification
Algorithm Specification
Design Products

Implementation

- 4-principles to consider:
  - Reduction of complexity
  - Anticipation of diversity
  - Structuring for validation/testing
  - Use of external standards

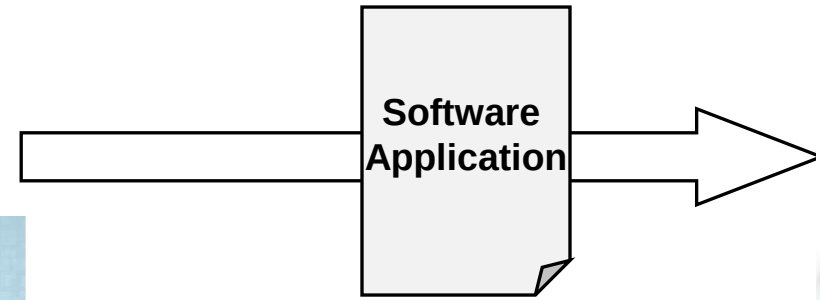
Verification and Validation (Testing)

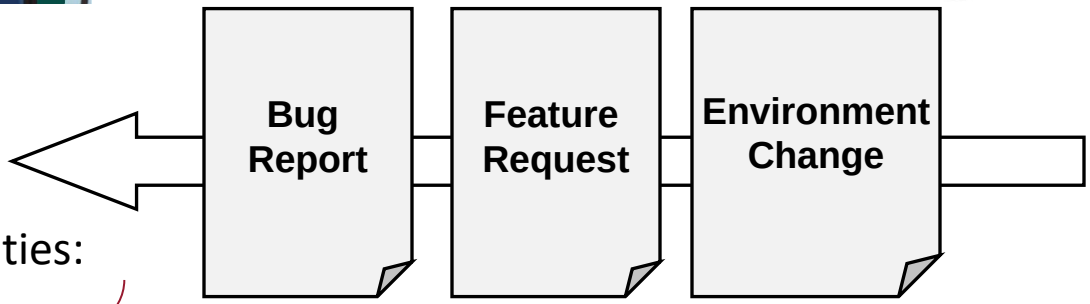
- Validation: did we build the right system?
- Verification : did we build the system right?

Unit Testing Integration Testing System Testing

And other testing techniques.

Maintenance





Maintenance activities:

- Corrective maintenance
- Perfective maintenance
- Adaptive maintenance

Regression Testing

The Software Lifecycle

- Software lifecycle is a series of phases through which software is produced:
  - from conception to end-of-life
  - can take months or years to complete
- Goals of each phase:
  - mark out a clear set of steps to perform
  - produce a tangible item
  - allow for review of work
  - specify actions to perform in the next phase

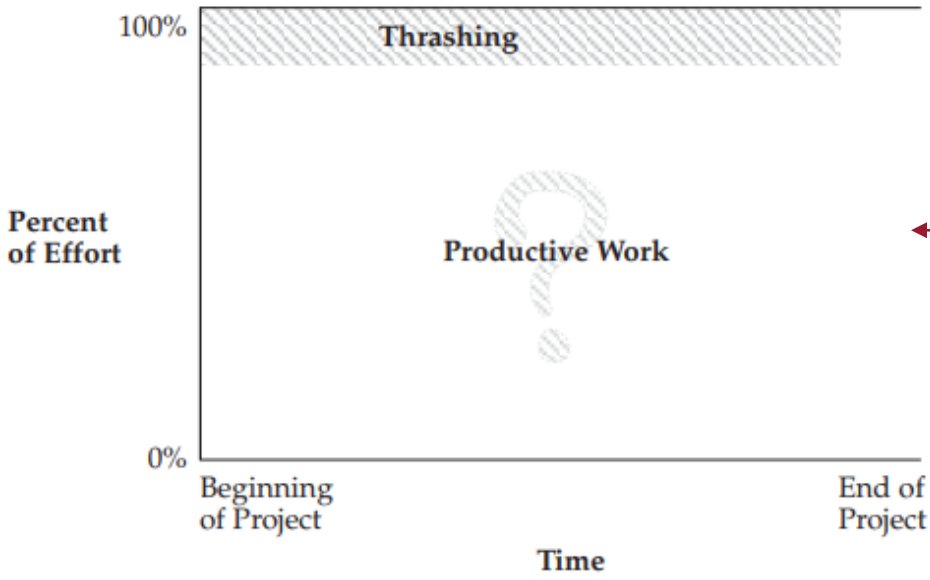
Life without software process

- Advantages:
  - nothing to learn or plan!
  - work on whatever is interesting, ignore the rest.
- Disadvantages:
  - code may not match user's needs (no requirements!)
  - not clear when to start or stop doing each task
  - may ignore some important tasks (testing, design)
  - scales poorly to multiple people
  - hard to review or evaluate one's work
  - code was not planned for modification, not flexible

Life with software process

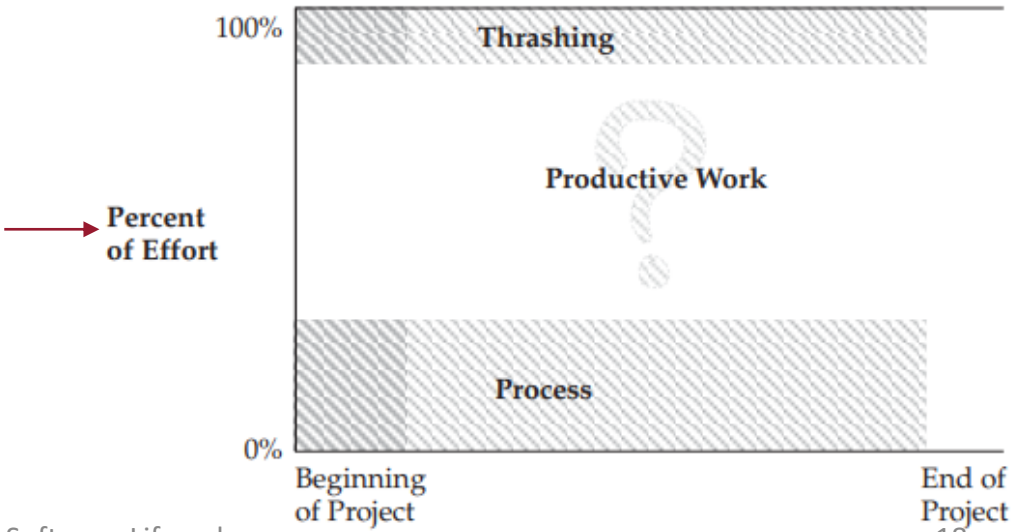
- Advantages:
  - Provides structure in which to work
  - Forces you to think about the big picture and to follow steps to reach it
  - Without it, you may make decisions that are locally optimal but globally misdirected
  - It is a management tool
- Disadvantages:
  - Can lead to compromises and artificial constraints
  - Risk of over-emphasizing process rather than the product itself!

Software Development



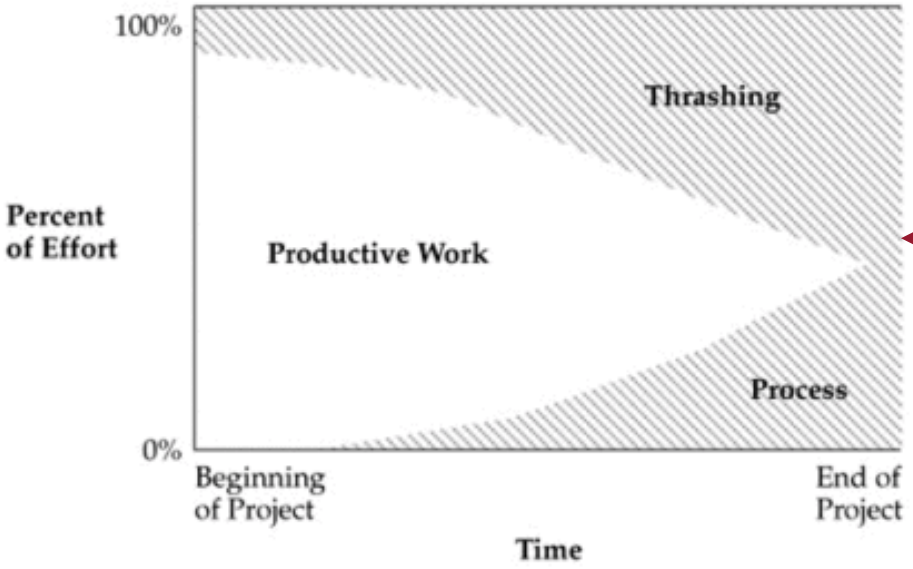
Mistaken perception that ignoring process increases the proportion of productive work on projects

Mistaken perception that an attention to process will decrease the proportion of productive work. (Process is seen as pure overhead.)



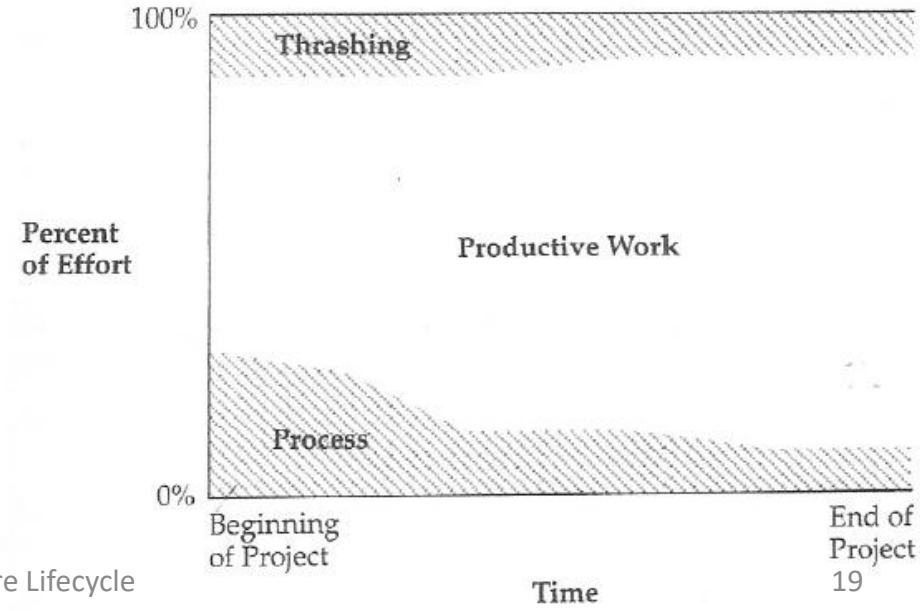
Survival Guide, McConnell, p. 25-26

Software Development



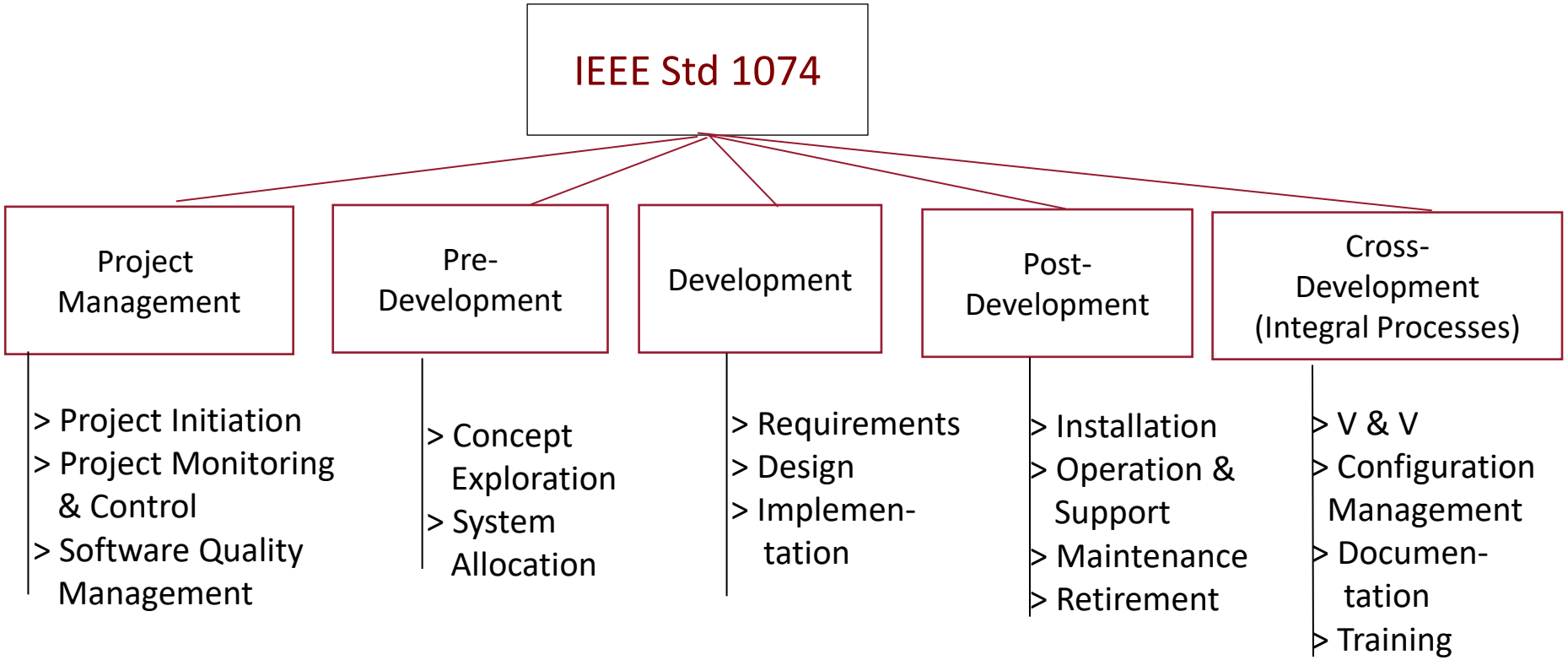
Project with little attention to process

Project with early attention to process



Survival Guide, McConnell, p. 25-26

IEEE Std 1074: Standard for Software Life Cycle Activities



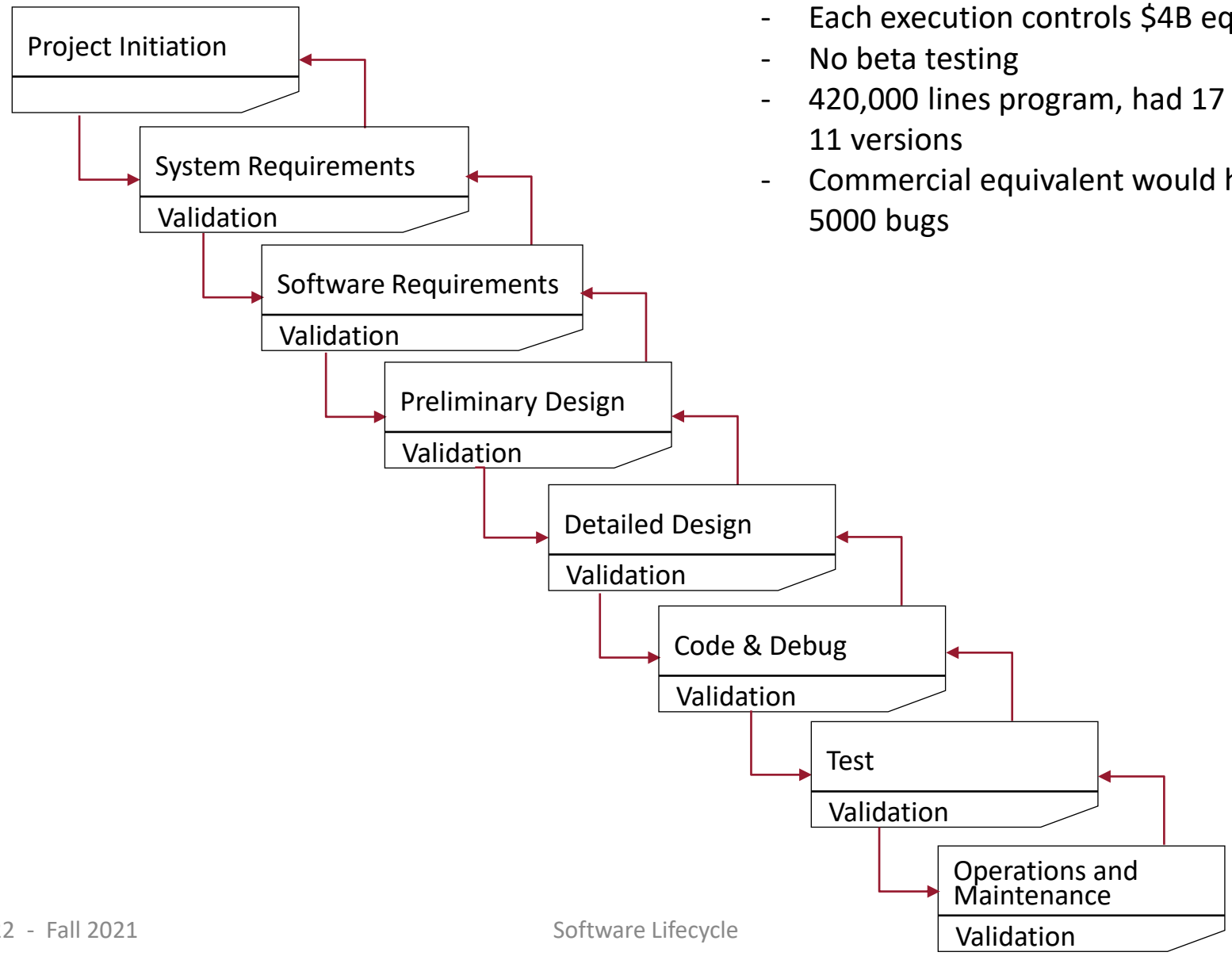
Some lifecycle models

- Waterfall Model:
  - perform each phase in linear order
- Spiral Model:
  - figure out riskiest things first
- Evolutionary Prototyping :
  - build initial requirement specs or several releases, then design-and-code each in sequence
  - do the next easiest thing that could possibly lead to feedback
- Rational Unified Process:
  - UML based, adaptable process framework, intended to be tailored according to the project needs.
- Agile Process:
  - iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end

Software Process Control Variables

- Control variables in a software project
  - Time
  - Quality
  - Scope (features)
- If you try to fix all three then the hardest to measure (i.e., quality) will suffer
- Management team control the first two and the development team controls the third variable

Waterfall Model



Waterfall Example at NASA

- Space shuttle control software
  - Each execution controls $4B equipment
  - No beta testing
  - 420,000 lines program, had 17 errors in 11 versions
  - Commercial equivalent would have 5000 bugs

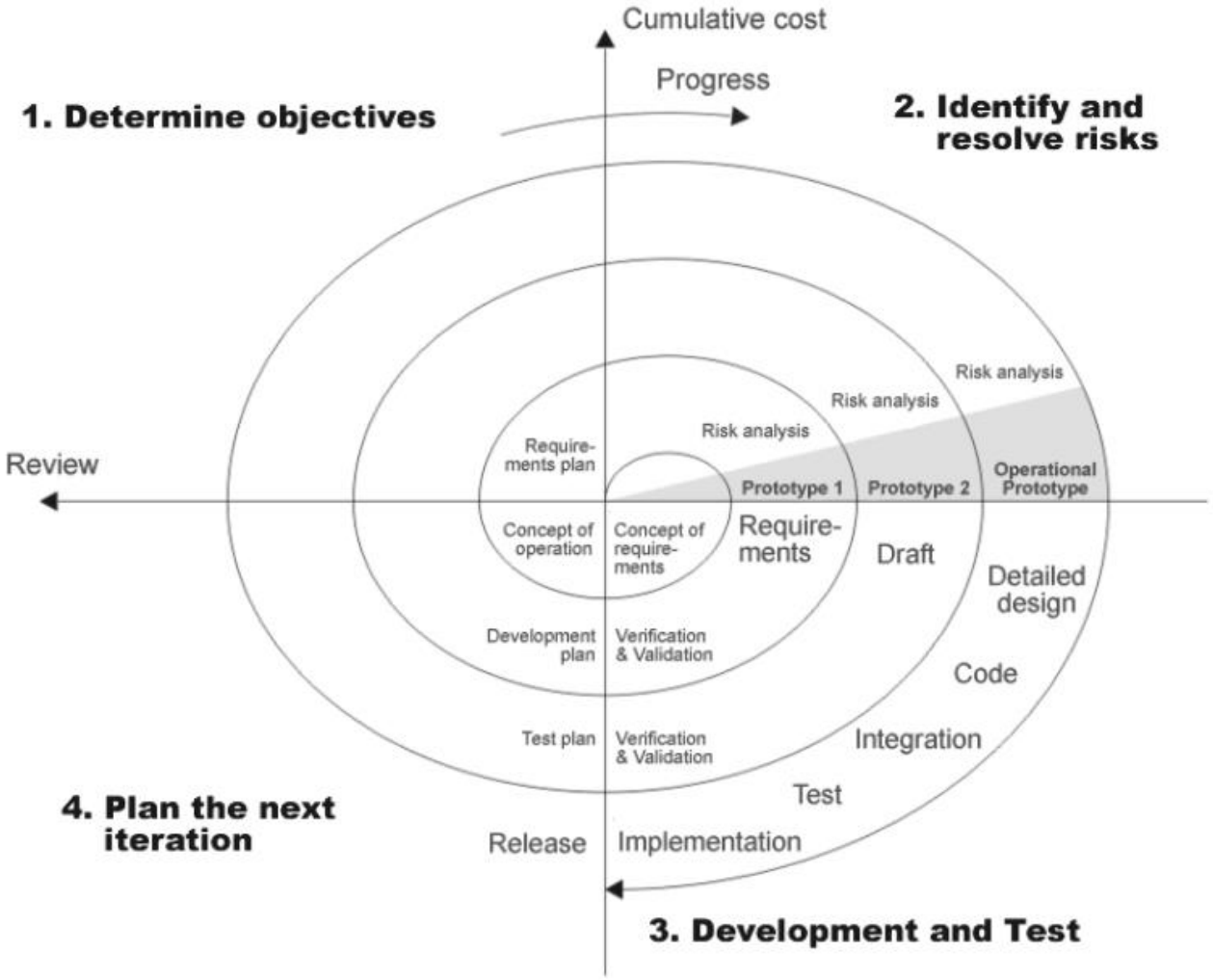
Waterfall Model Advantages

- Suitable for projects that are very well understood but complex
  - Tackles all planning upfront
  - The ideal of no midstream changes equates to an efficient software development process
- Supports inexperienced teams
  - Orderly, easy-to-follow sequential model
  - Reviews at each stage determine if the product is ready to advance

Waterfall Model Disadvantages

- Requires a lot of planning up front (not always easy)
  - assumes requirements will be clear and well-understood
  - a third of the effort before coding starts
- Rigid, linear; not adaptable to change in the product
  - costly to "swim upstream" back to a previous phase
  - E.g., NASA Project: Change to add GPS support (1% of code = 7k lines). Spec for the change is 2500 pages ! – Total spec is 40,000 pages
- No sense of progress until the very end
  - no code to show until almost done
- Integration occurs at the very end
  - defies “integrate early and often” rule
  - solutions are inflexible, no feedback until end
- Delivered product may not match customer needs
  - phase reviews are massive affairs
  - inertia means change is costly

Spiral Model (risk oriented)



Spiral Model Advantages

- Use Spiral:
  - When project is large and medium to high risk
  - When requirements are unclear and complex
  - When changes may require at any time
  - When risk and costs evaluation is important
  - When creation of a prototype is applicable
  - When releases are required to be frequent

- Risk reduction: Always addresses the biggest risk first
- Provides early indication of unforeseen problems
- Accommodates change
- Software is produced early (can get early feedback from customer)

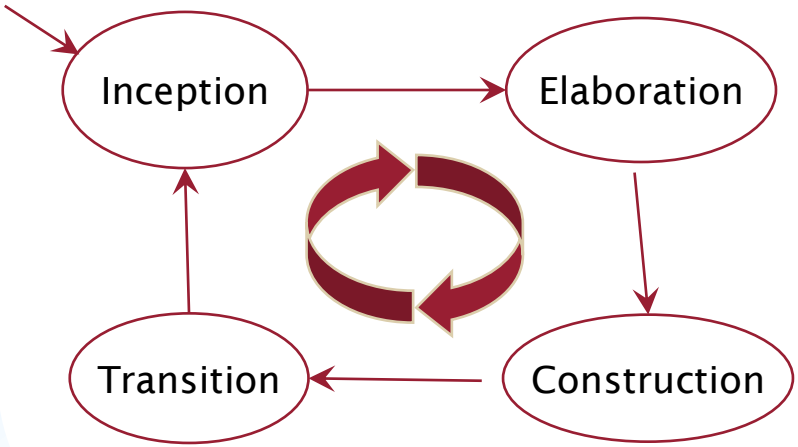
Spiral Model Disadvantages

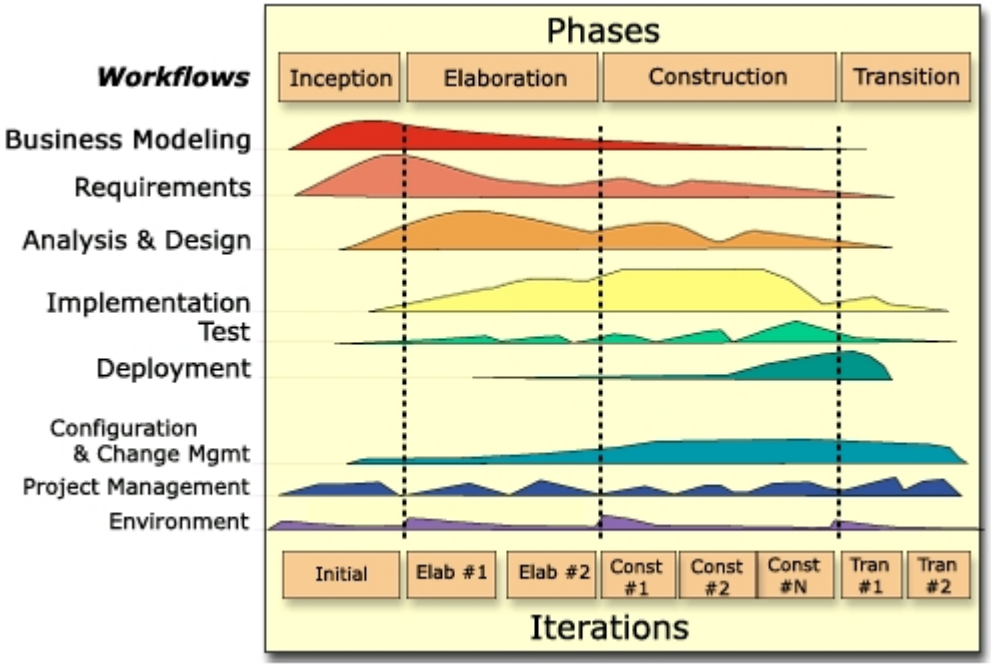
- Complex. Will not work well for small scale projects.
  - A lot of planning and management
  - Frequent changes of task
  - Requires customer and contract flexibility
- Needs specific expertise
  - Developers must be able to assess risk

Rational Unified Process

This is the most widely known methodology that embraces UML

- Designed to be adaptable
- Adapts to the project needs
- Use-case-driven development
- Architecture-centric process





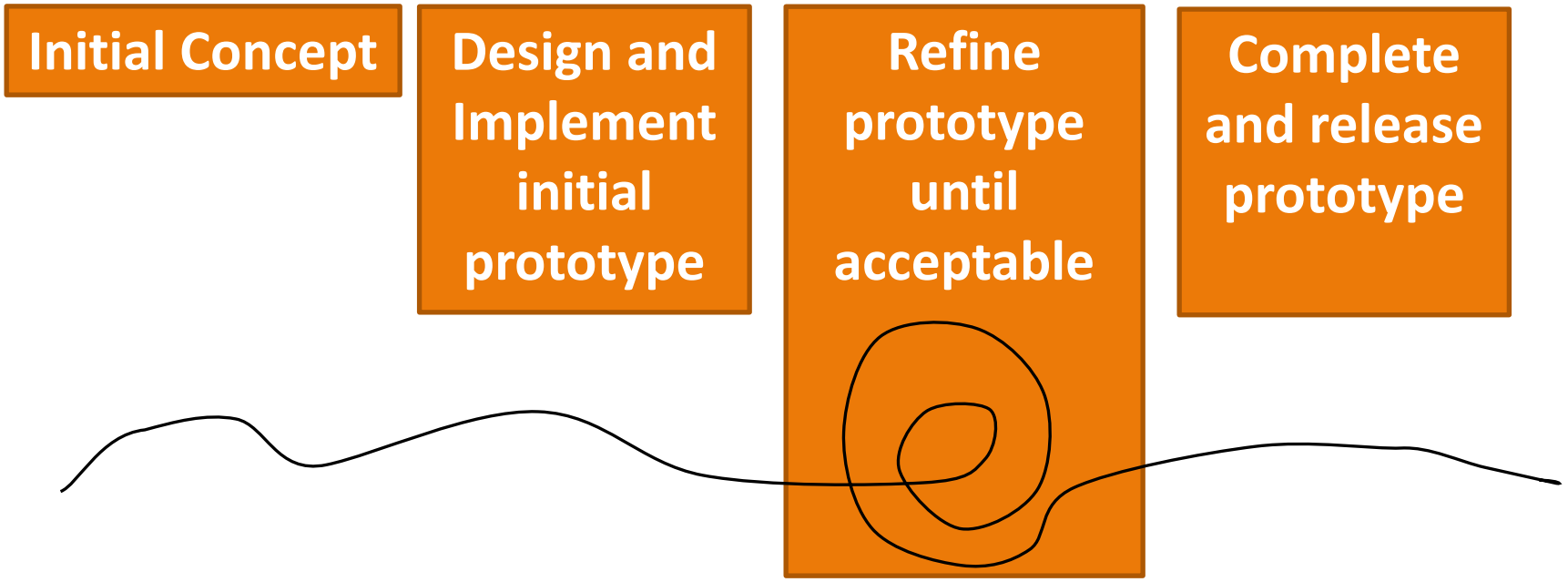
Rational Unified Process Advantages

- Regular feedback from and to customers and stakeholders
- Efficient use of resources
- You deliver exactly what the customer wants
- Issues are discovered early in your project
- Improved control
- Improved risk management

Rational Unified Process Disadvantages

- Complex:
  - The process can be too complex to implement
  - Development can get out of control
  - It is a heavy weight process
  - You need an expert to fully adopt this process

Evolutionary Prototyping



Very practical, widely used, and successful!

- Develop a skeleton system and evolve it for delivery
- Requirements are not known ahead of time. Discovered by feedback.

Evolutionary Prototyping Advantages

- Immediate user feedback:
  - Steady signs of progress build customer confidence
  - Useful when requirements are unknown or changing
  - Participatory design / useful feedback loops
  - Addresses risks early

Evolutionary Prototyping Disadvantages

- Requires close customer involvement
- Assumes user's initial spec is flexible
- Problems with planning
  - especially if the developers are inexperienced
  - feature creep, major design decisions, use of time, etc.
  - hard to estimate completion schedule or feature set
  - unclear how many iterations will be needed to finish
- Integration problems
  - fails for separate pieces that must then be integrated
  - bridging; new software trying to gradually replace old
- Temporary fixes become permanent constraints

Comparing Process Models

- **Waterfall:**

Pros:
1. It is easy to understand and plan
2. It works for well-understood projects
3. Analysis and testing are straightforward

Cons:
1. It does not accommodate change well
2. Testing occurs late in the process
3. Customer approval is at the end

- **Spiral:**

Pros:
1. Continuous customer involvement
2. Development risks are managed
3. Suitable for large, complex projects
4. It works well for extensible products

Cons:
1. Risk analysis failures can doom the project
2. Project may be hard to manage
3. Requires an expert development team

- **Rational Unified Process:**

Pros:
1. Quality documentation emphasized
2. Continuous customer involvement
3. Accommodates requirements changes
4. Works well for maintenance projects

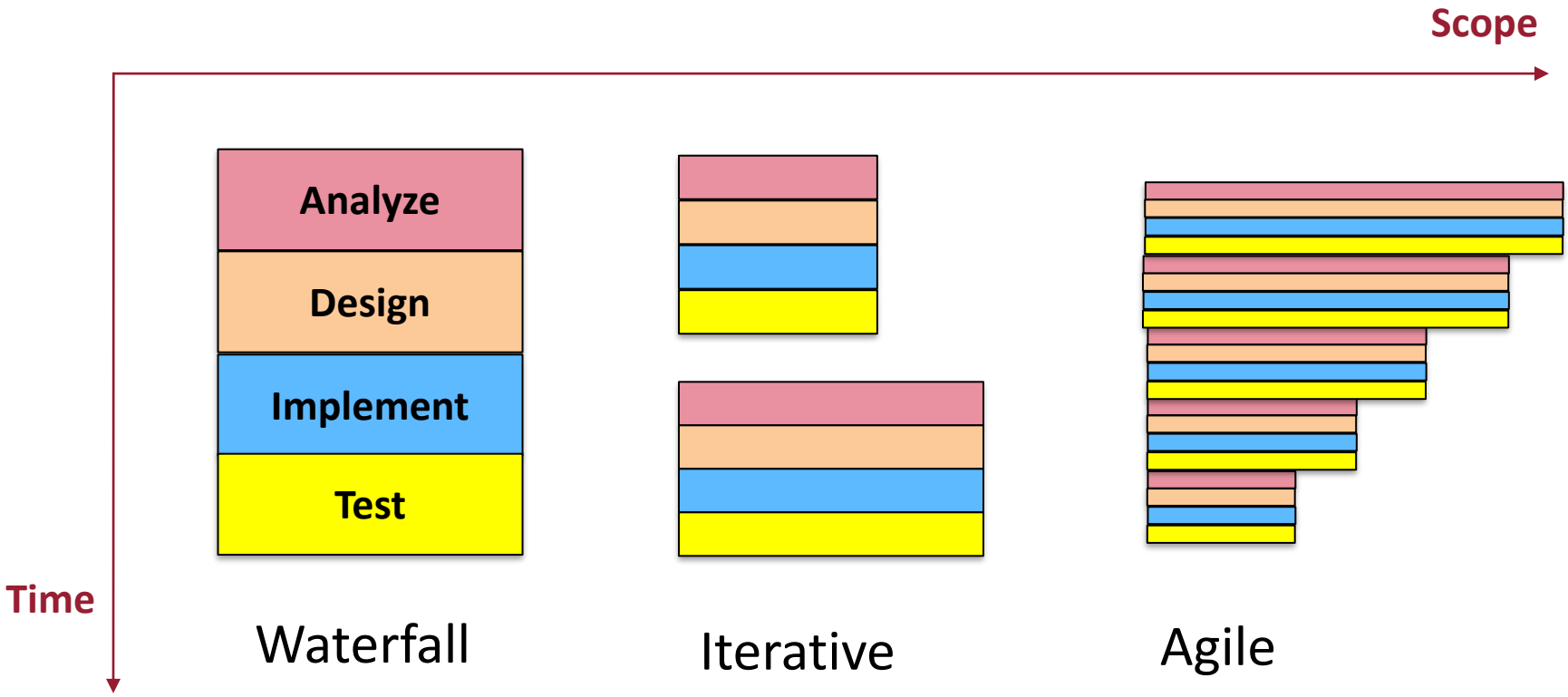
Cons:
1. Use cases are not always precise
2. Tricky software increment integration
3. Overlapping phases can cause problems
4. Requires expert development team

- **Evolutionary Prototyping:**

Pros:
1. Useful when requirements are unknown or changing
2. Customer is involved early and often
3. Works well for small projects
4. Reduced likelihood of product rejection

Cons:
1. Customer involvement may cause delays
2. Temptation to "ship" a prototype
3. Work lost in a throwaway prototype
4. Hard to plan and manage

Agile Processes



- Agile process is an iterative and incremental process with short iterations

Why are there so many models?

- The choice of a model depends on the project circumstances and requirements.
- A good choice of a model can result in a vastly more productive environment than a bad choice.
- A cocktail of models is frequently used in practice to get the best of all worlds. Models are often combined or tailored to environment.

What's the best model?

- Consider
  - The task at hand
  - Risk management
  - Quality / cost control
  - Predictability; Visibility of progress
  - Customer involvement and feedback

Time
Quality
Scope (features)

It is not always possible to optimize all three at the same time.

What's the best model for …

- A system to control anti-lock braking in a car
- A learning management system similar to Blackboard Learn or Canvas.
- An new interactive system that allows airline passengers to quickly find replacement flight times (for missed or bumped reservations) from terminals installed at airports
- A mobile app for finding friends