Software Development Lifecycle

JD Kilgallin

CPSC:480

08/29/22

Pressman Ch 2-4

His forehead says "Do not develop my app" -Morty Smith, Rick and Morty. Adult Swim. 2019.

Learning Objectives

- Software development process models

- Phases of development and their elements

- Common and historical software engineering methodologies

- Evolution of software systems

How does software get developed?

- A product is conceived to meet a business need.
- Stakeholders communicate with a software engineering team to design and develop (the first version of) the product.
- Work is scoped, broken down into tasks, and assigned to developers.
- Code is written, built, tested, documented, released, and deployed.
- Software starting here, from scratch, is called *greenfield* development.
- Process is repeated to create the next version, or the team shifts to a different product. Work using a pre-existing codebase is *brownfield.*
- The cycle of defining requirements, planning implementation, designing the product, implementing it, testing it, and releasing it for each iteration is the fundamental role of Software Engineering teams.

How does software development start?

- Initial planning may precede the first cycle to scope the entire project.
- Inception – Someone identifies a need for software to address a specific use case. May be software vendor executives deciding to launch a product based on perception of market need, or a customer seeking to hire a company to build a product.
- Elicitation – Product team works to define basic requirements.
- Elaboration – Requirement details filled in to understand exact needs.
- Negotiation – Budgets, timelines, and resource availability considered
- Specification – A full(ish) statement of the work to be performed.
- Validation – Specification is confirmed to meet customer needs.

Software development process

- A framework for the activities, actions, and tasks required to build high-quality software.

- The approach that is taken as software is engineered.

- A system in which tools and techniques are used to develop software.

- A series of predictable steps to deliver quality software on time.

- An iterative strategy for successful software release and evolution.
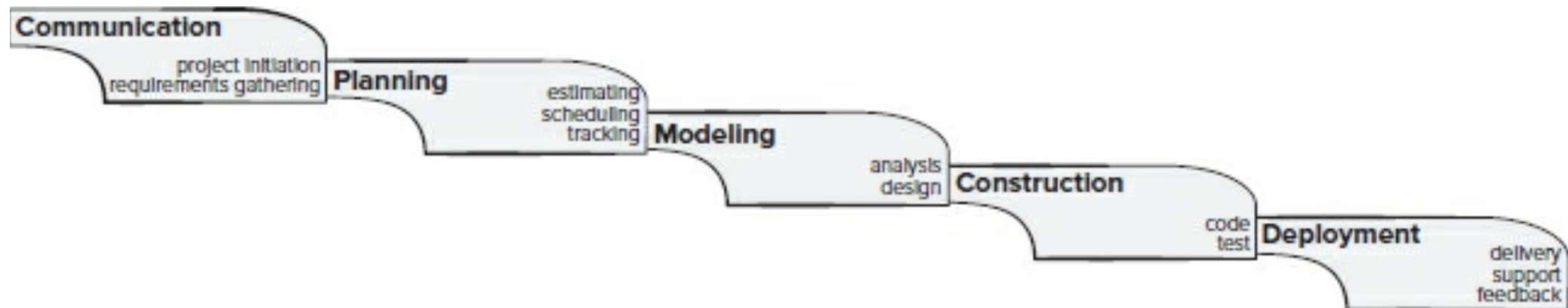
Software development activities

- Communication (Requirements) – Deciding what the software will do
- Planning – Deciding what needs to be done to achieve requirements, including allocation of time for product design/modeling
- Modelling (Product Design) – Deciding how the software will be built
- Construction
  - Implementation – Creating the software according to design
  - Validation (Testing, aka Quality Assurance (QA)) – Confirming the software meets requirements
- Release (Deployment) – Delivering software package to the consumer
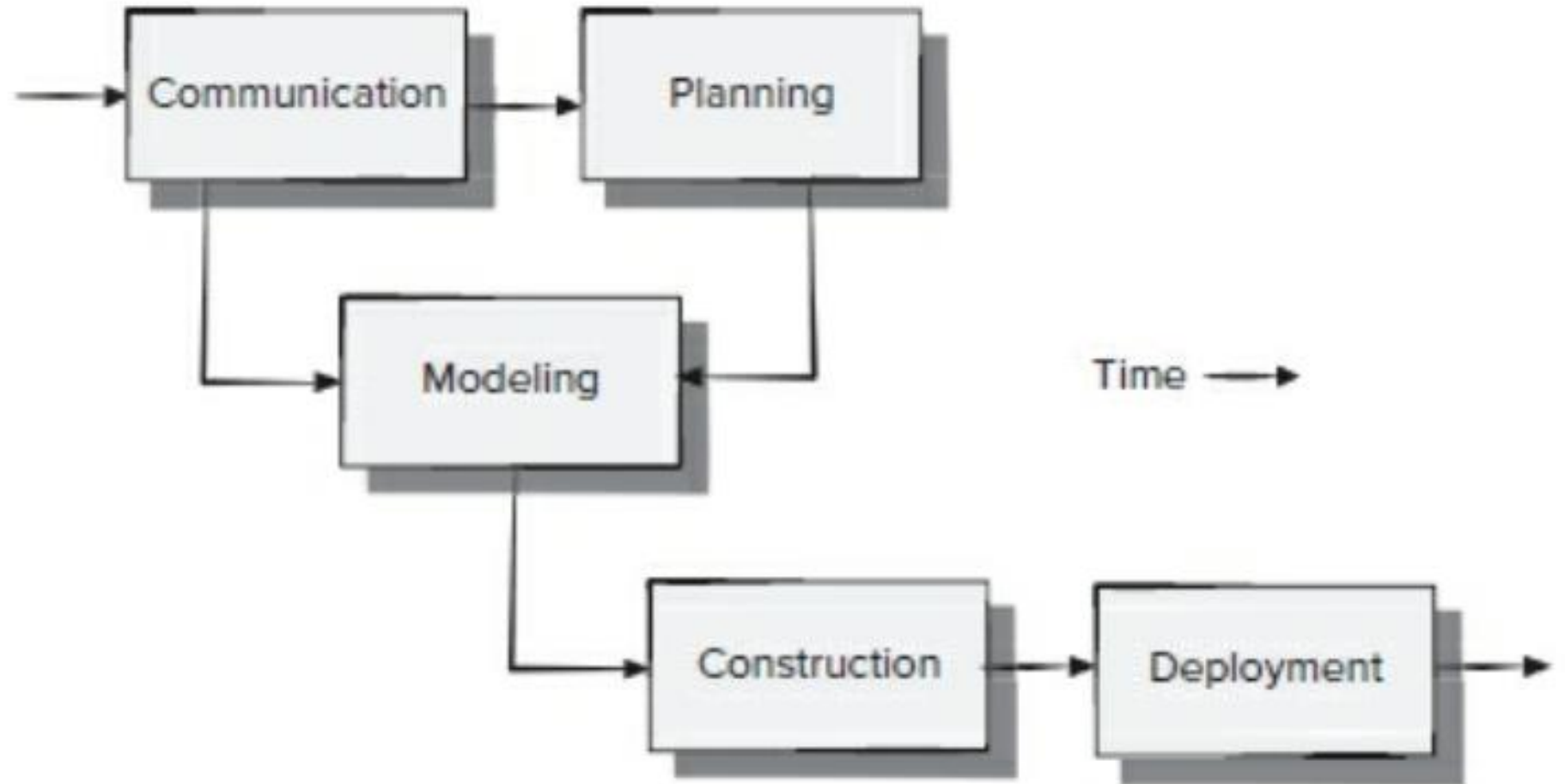
Waterfall model
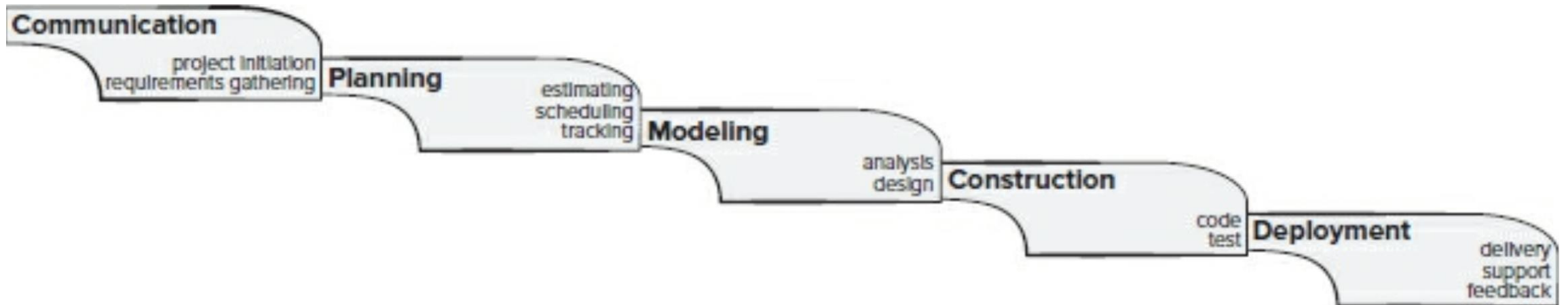
Waterfall model ~~pros and~~ cons

- 1970's approach
- Does not capture real projects and cannot be justified to stakeholders
- Leaves many contributors with nothing to do for much of the project
- Changing requirements significantly delay the entire project
- Customer feedback cannot be gathered until the end
- *Does not allow for feedback to be incorporated in the first place!*

Communication
project initiation
requirements gathering **Planning**
estimating
scheduling
tracking **Modeling**
analysis
design **Construction**
code
test **Deployment**
delivery
support
feedback

Parallel waterfall model
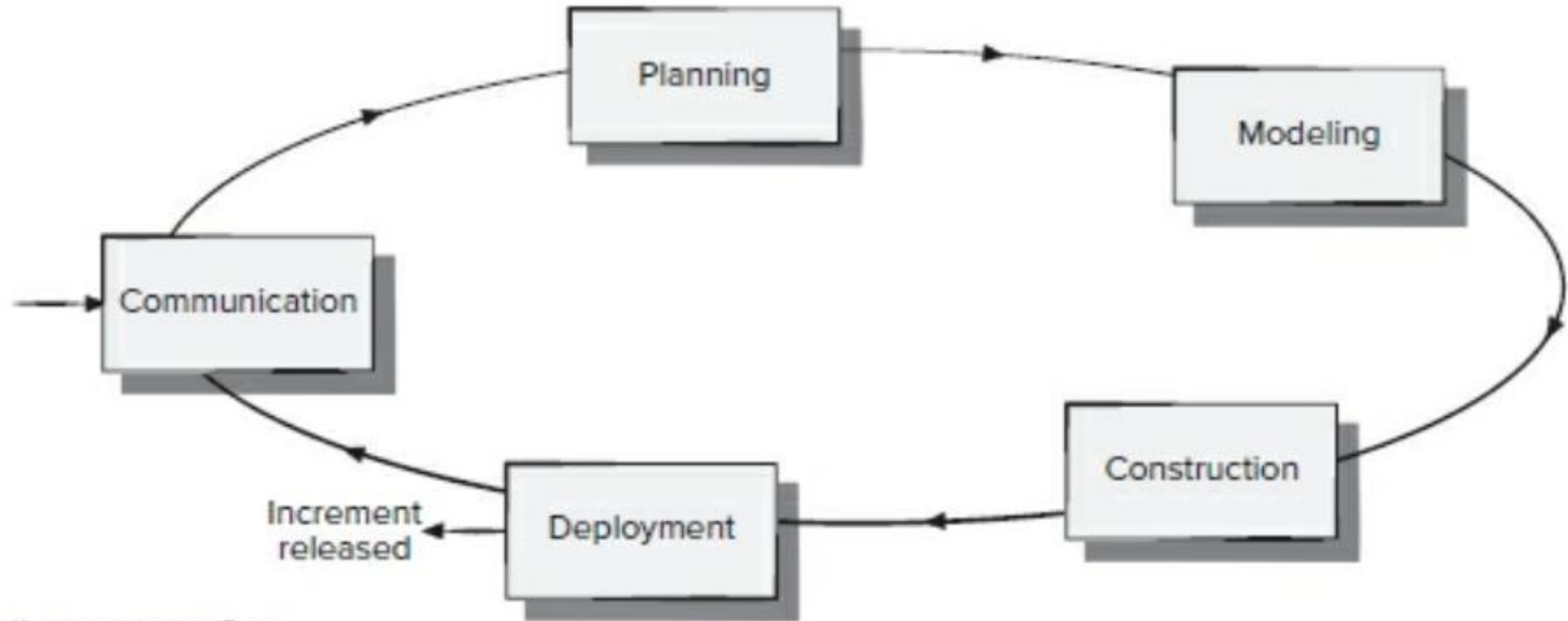

(d) Parallel process flow

Parallel waterfall pros and cons

- Less downtime: Contributors participate through most of the project
- Changing requirements no longer delay implementation of unrelated elements.
- However, customer feedback still cannot be gathered until the end, and cannot be effectively incorporated.
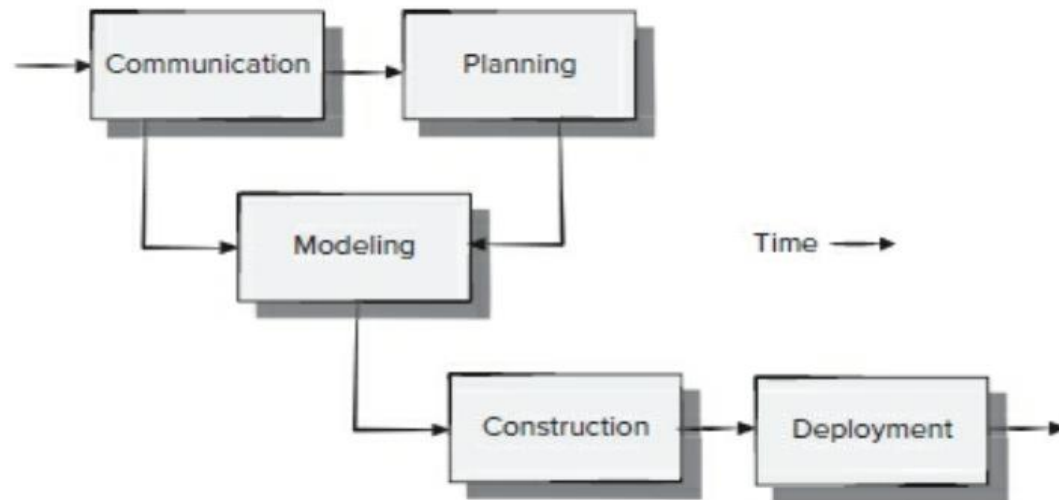
(d) Parallel process flow

Prototyping model

(c) Evolutionary process flow

Prototyping model pros and cons

- Can be used with poorly-defined initial requirements
- Customer feedback can be gathered and used quickly
- Direction of product can shift easily
- Features can be implemented in order of priority

- Working to quickly develop a prototype leads to compromising on quality of design and of implementation – *technical debt*
- Customers see what appears to be working software without understanding of technical debt and are less willing to allow for design and implementation improvements.
- Little testing: software may function poorly outside of core use cases.
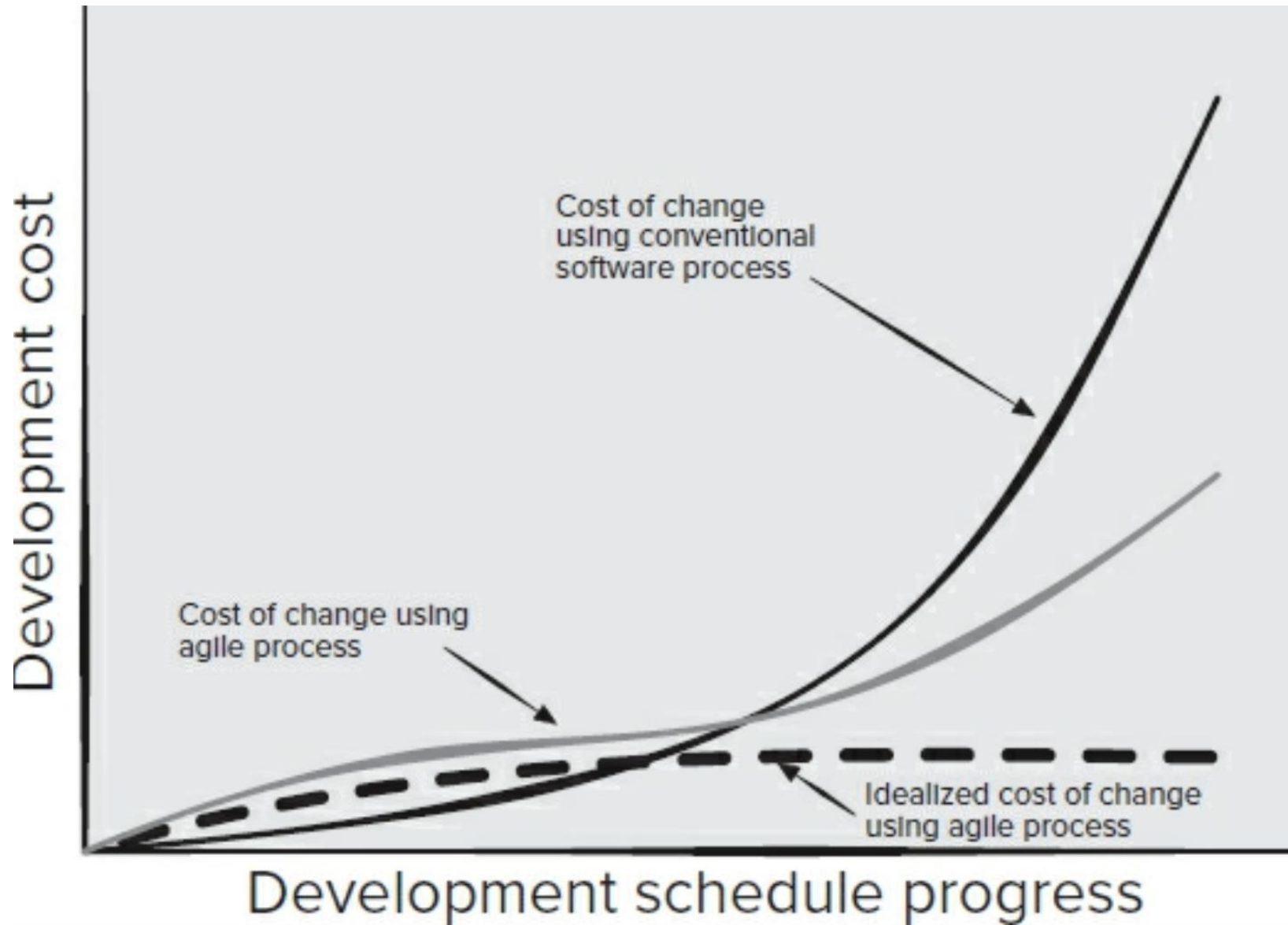
Iterative model

- Reduces likelihood of significant technical debt.

- Increased time spent on requirements, planning, and design upfront reduces cost of change without holding up construction of features ready to be implemented.

- Starting to look like a simplified version of a realistic process.
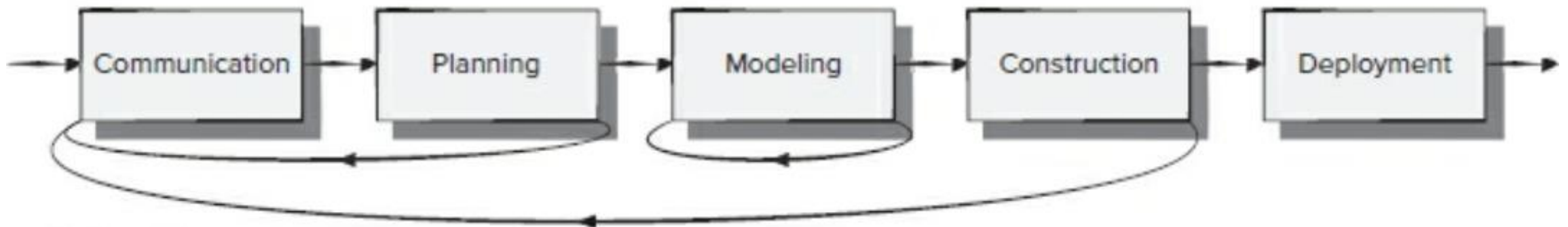
(b) Iterative process flow

Agile model

- Philosophy and guidelines to reduce complexity and cost of development.

- Variations widely used in the field today. Popular with developers.

- Central concept is that traditional models fail to capture human factors of software development processes.

- Acknowledges difficulty of predicting how requirements will evolve and that process phases cannot happen disjointly, even in parallel.

- Aims to reduce the cost of change throughout the product lifetime.

- Covers software team organization, work artifacts, & unpredictability.

Cost of change using conventional software process

Cost of change using agile process

Idealized cost of change using agile process

Development cost
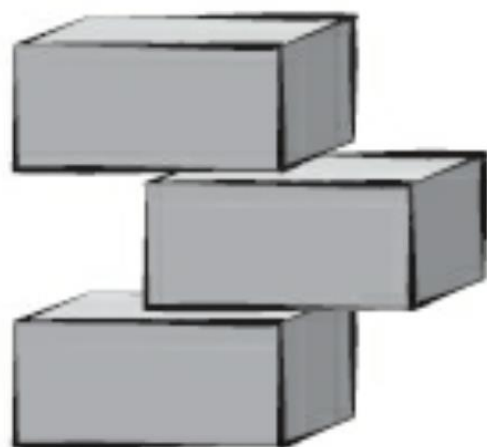
Development schedule progress

Agile Alliance Principles (paraphrased)

- Highest priority is customer satisfaction.
- Welcome changing requirements for competitive advantage.
- Deliver working software as frequently as possible (down to 2 weeks).
- Business and development staff must work together.
- Get work done by giving motivated people a conducive environment.
- Prefer face-to-face communication.
- Working software is the primary measure of progress.
- Promote an indefinitely-sustainable pace of development.
- Continuously pay attention to technical excellence and good design.
- Simplicity – maximizing the amount of work not done – is essential.
- The best designs emerge from self-organizing teams.
- Routinely reflect on how to become more effective and adjust accordingly.

Agile processes – (somewhat) common features

- Scrum meetings – Short "stand-up" meetings at the beginning of each day to describe previous day's work, plans for that day, and obstacles.
- Product backlog – A list of work tasks to be prioritized and assigned.
- User stories – Capturing requirements in the form "As a ___, I want to ___"
- Product owner – Responsible for backlog item definition and priority.
- Sprints – A short, 2-4-week block in which selected work items and bug fixes are completed by developers, resulting in a software increment.
- Sprint review meeting – Demonstrate features developed that sprint.
- Sprint planning – Work to be done next sprint is estimated and assigned. No new work *should* be added to the sprint once it has begun.
- Pair programming (XP) – Two developers, one computer for each item.
- Task (Kanban) board – Table showing current status on each work item.

Product Backlog:
Prioritized product features desired by the customer

Scrum:
15-minute daily meeting.

every 24 hours

30 days

Sprint Backlog:
Feature(s) assigned to sprint

Backlog items expanded by team

New functionality is demonstrated at end of sprint

simple design
CRC cards

spike solutions
prototypes

user stories
values
acceptance test criteria
iteration plan

design

planning

refactoring

coding

testing

pair programming

Release

software increment
project velocity computed

acceptance testing

Unit test
continuous integration

Backlog	Selected	Analysis		Development		Testing	Done
		Doing	Done	Doing	Done		

Expedite Lane

398 Cancel order form

👤 **Christie Church**   💬 1   [Phone ✕] [Web ✕] [+]   💾 Save & Close ▾   👁 Follow   •••

State	🔵 Active		Area	Fabrikam Agile Project	Updated 8/19/2016
Reason	Implementation started		Iteration	Fabrikam Agile Project	

Details | 🕘 | 🔗 (4) | 📎

Description ⛶ ⌄

B *I* U̲ A̶ 🔗 ✂🔗 ☰ 1☰ ⇤ ⇥ 🖼

Provide a **cancellation order** form similar to the screen shown. See the attached storyboard for details.

Order details here
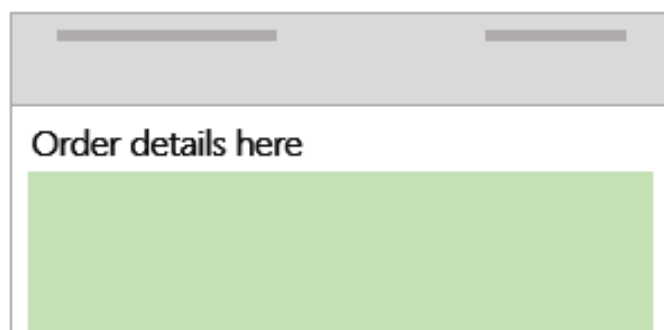
Acceptance Criteria ⛶ ⌄

Discussion ⛶ ⌄

Add a comment. Use # to link a work item or @ to mention a person

Helena commented *2 months ago*
Reassigning this to Christie

Planning ⌄

Story Points

Priority
2

Risk

Classification ⌄

Value area
Business

Development ⛶ ⌄

➕ Add link

Development hasn't started on this item.
Create a new branch

Related Work ⛶ ⌄

➕ Add link ⌄

Parent
340 Improve User Experience
Updated 7/31/2014, 🔵 Active

Child
466 Develop standards guid...
Updated 7/12/2016, 🔵 Active

Related
468 Customer account history
Updated 8/19/2016, 🔵 Active

What happens after software is developed

- Final release candidate built for internal and beta testing.
- Final documentation and release notes compiled.
- Marketing materials announcing new features are released.
- Sales, operations, & partner-company staff is trained on new version.
- When all required parties sign off on release, software, documentation, and related materials (e.g. containers) are packaged and distributed through applicable channels.
- Support and operations staff begin working with the new version in production environments.
- Maintenance of this version and development of next version begins.
- Release retrospective identifies areas of process improvement and computes development *velocity* to assist with future time estimates.

Software maintenance

- Occurs through the entire life of the product. More expensive than the original development due to the long duration.

- *Perfective maintenance* occurs to proactively improve product performance, existing functionality, architecture, and documentation. This is often planned before the version to be maintained is released.

- *Corrective maintenance* occurs in reaction to a bug or defect whose existence or impact is only identified after a release is deployed.

- *Adaptive maintenance* occurs in reaction to changes in the environment in which the software needs to run (e.g. new version of underlying database server has new security requirements).

- *Preventive maintenance* occurs to proactively fix potential issues before they cause impact to customers (e.g. developers, testers, or users find an exploitable, but not yet exploited, weakness in application security).
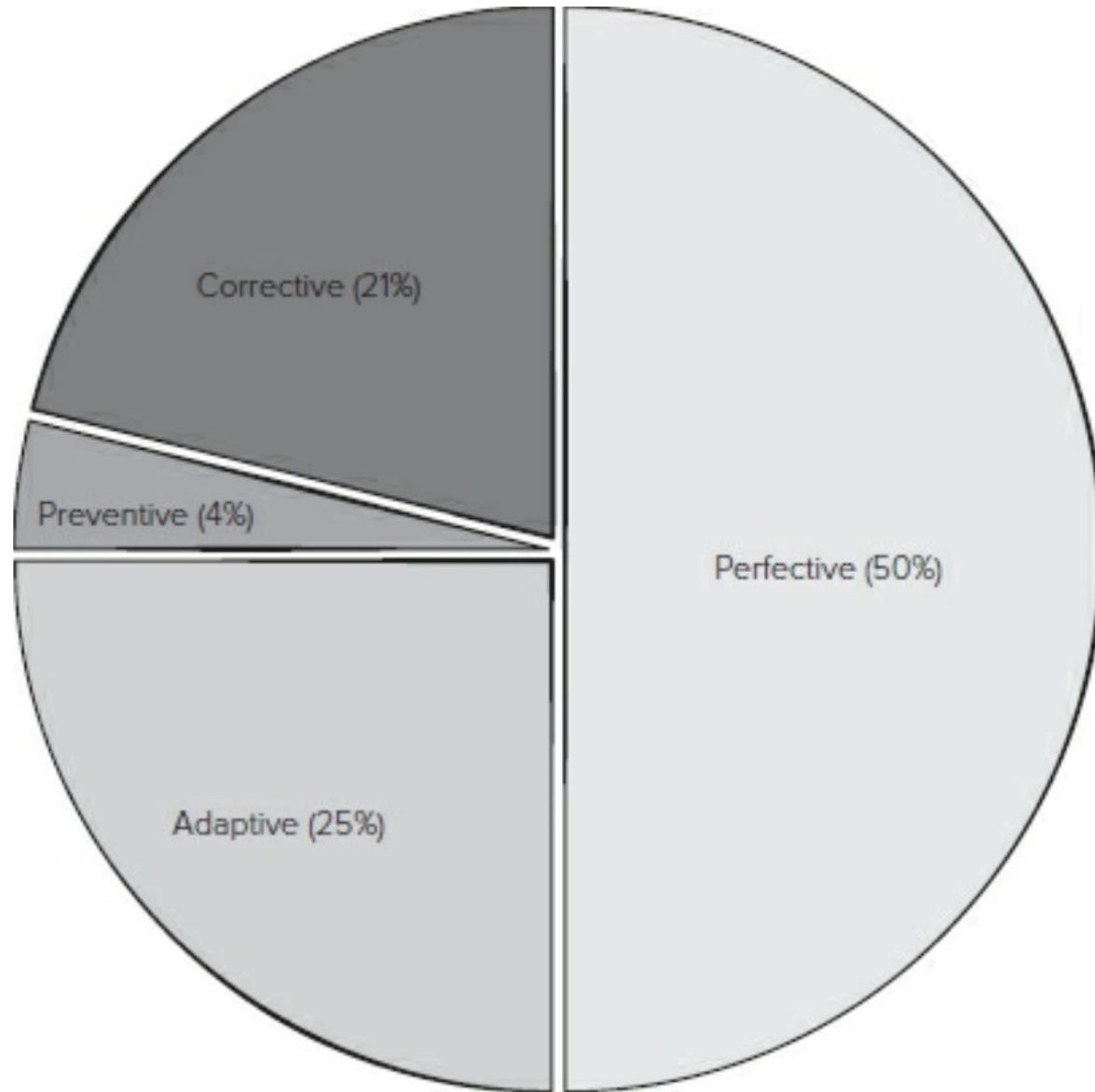
Evolution

- Product development rarely stops with one released version.
- Timelines prevent all desired features from being developed at once.
- Additional versions are frequently planned before the first version is released.
- Some current software systems have been around for *decades*
  - Voyager 2 probe, begun 1972
  - MOCAS US DoD contract management, 1958
  - GNU Emacs, 1984
  - C Pre-processor, 1978
- Even after the last planned release, new versions of *legacy software* are sometimes developed later.

Lehman's 8 Laws of Software Evolution

- Recall: An *E*-program is written to perform some real-world activity.

- "Continuing Change" — An E-type system must be continually adapted or it becomes progressively less satisfactory.

- "Increasing Complexity" — As an E-type system evolves, its complexity increases unless work is done to maintain or reduce it.

- "Self-Regulation" — E-type system evolution processes are self-regulating, with the distribution of product and process measures close to normal.

Lehman's 8 Laws of Software Evolution

- "Conservation of Organizational Stability" — The average effective global activity rate in an evolving E-type system is invariant over the product's lifetime.

- "Conservation of Familiarity" — As an E-type system evolves, all associated with it must maintain mastery of its content and behavior to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.

Lehman's 8 Laws of Software Evolution

- "Continuing Growth" — The functional content of an E-type system must be continually increased to maintain user satisfaction over its lifetime.

- "Declining Quality" — The quality of an E-type system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes.

- "Feedback System" — E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

```
                                      Preliminary
         Requirements          Architectural Design
         Engineering
                                                        Estimate Required
                                                        Project Resources

    Project
  Conception
                                                                    Construct 1st
                                                                    Prototype
              Construct
              Next Prototype
                                                    Evaluate
                                                    Prototype

                         Prototype Becomes
                         Software Release

    Redefine
    Scope
                              Maintain                    Go, No-Go
                              Software                    Decision

                                                                         End
                         Evolve                                          Project
                         System
```
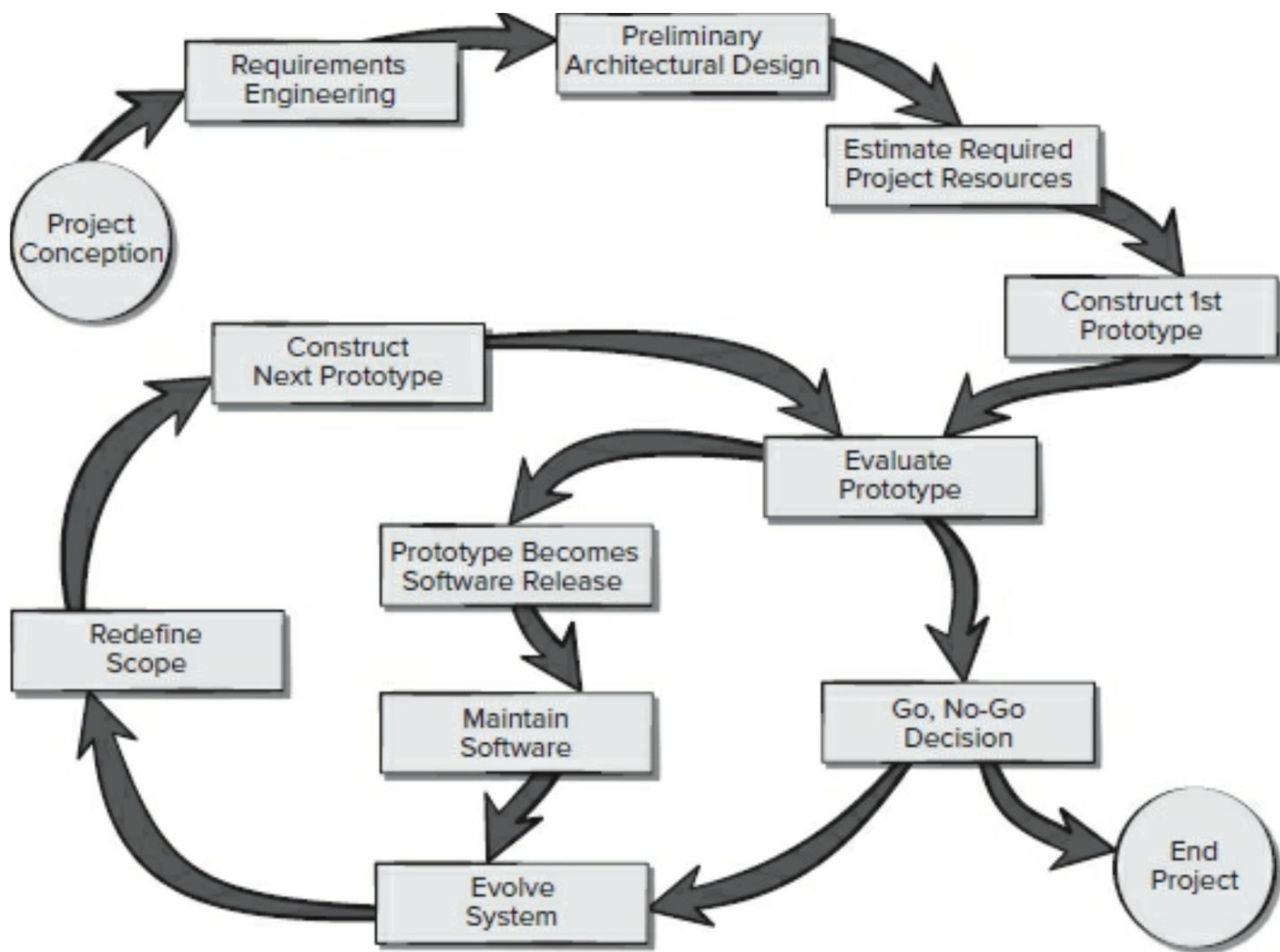
DevOps

- Merging the disciplines of software *Development* and *Operation.*

- Software development processes – even agile ones – do not capture the steps required in deploying and using the software*.*

- Deployment of modern, complex *cloud-native* applications running in datacenters require extensive engineering of the operating environment itself.

- Lack of coordination between software developers and operators leads to failures and inefficiencies, increasing cost of business.

DevOps – Infrastructure as code

- The environment in which software runs in production is not well-reflected by the environment in which it is developed and tested.

- Working configurations cannot easily be reproduced.

- Specifying the environment configuration in the same way as program code allows known working configurations to be deployed, and effects of environment change can be predicted.

- Infrastructure-as-code allows number of instances of software components to be scaled quickly to meet demand (e.g. Prime Day).

- Amenable to modern cloud architectures, where functionality is achieved by composing independent *microservices* running as *containers* (virtualized hardware) in a datacenter.

- i.e. "it works on my machine!" "Then let's ship your machine"

DevOps – CI/CD

- When independent parts of large, complex systems are updated in isolation, incompatibilities can develop. This can happen even within a single application with many features.

- The traditional process of *upgrading* deployed software can be risky and cumbersome, so updates of production software are infrequent and may not be used. Updates may cause unintended and irreversible effects.

- Emergency updates (e.g. security patches) disrupt development teams.

- Customers must wait extended periods on needed software updates.

- DevOps stresses *Continuous Integration* and *Continuous Deployment* to mitigate the impact of these issues. Frequent, minimal changes reduce friction between software components users.

- In a microservice architecture, new versions of components can be deployed alongside old versions to identify unintended effects with minimal customer impact.

Plan → Code → Build → Test → Integrate → Deploy → Operate → Monitor
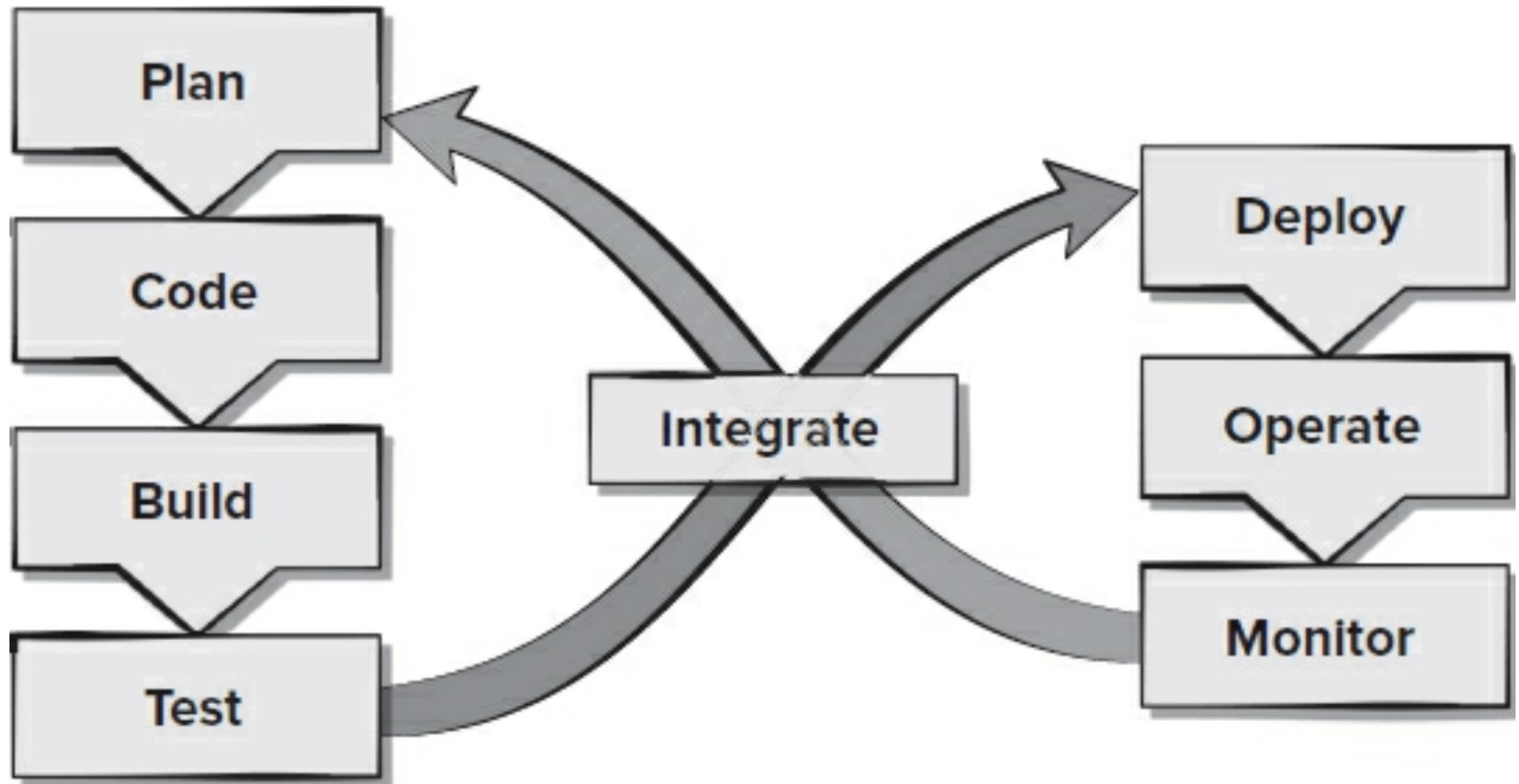
References

- Managing the Development of Large Software Systems. Winston Royce. Aug 1970. Proceedings of IEEE WESCON.
- The New Methodology. Martin Fowler. Dec 2005. MartinFowler.com.
- 12 principles behind the agile manifesto. Jul 2016. Agile Alliance.
- What is the Oldest Computer Program Still in Use? Glenn Fleishman. Aug 2015. MIT Technology Review.
- On understanding laws, evolution, and conservation in the large-program life cycle. Meir Lehman. Dec 1980. Journal of Systems and Software.
- *Reading for next lecture: Pressman Ch 5, 24,* Problems - LeetCode