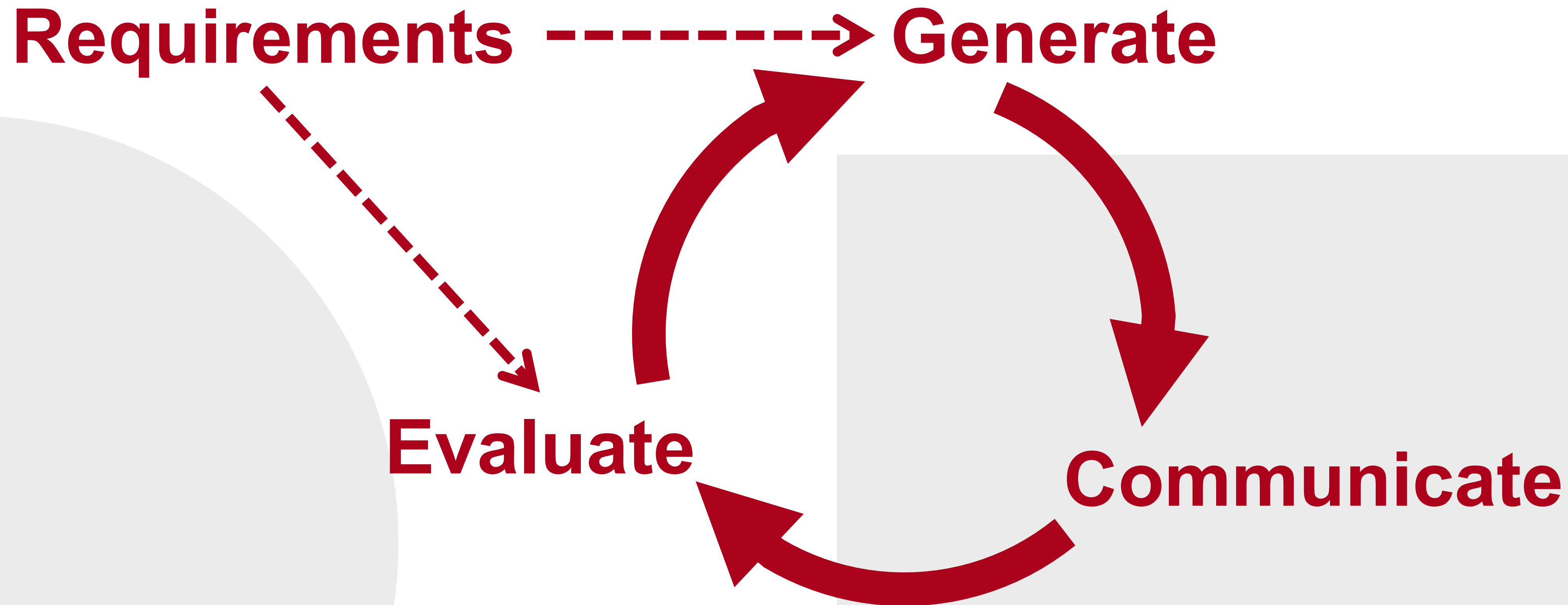
17-723: Designing Large-scale Software Systems

Generating Design Alternatives

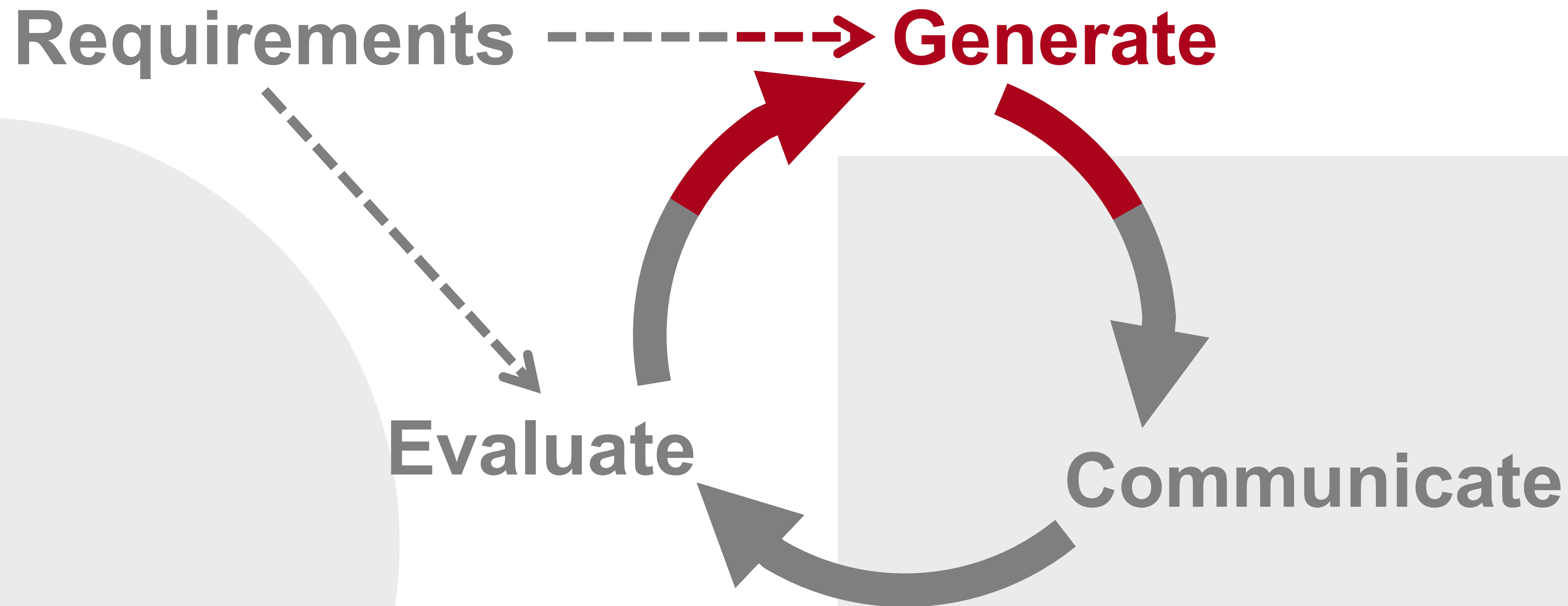
Carnegie Mellon University

Recall – The GCE Paradigm



Carnegie Mellon University

This Lecture - Generate



Carnegie Mellon University

This Lecture - Generate

- How to come up with a solution?
- How to refine a solution?
- How to solve a complex design problem?

Carnegie Mellon University

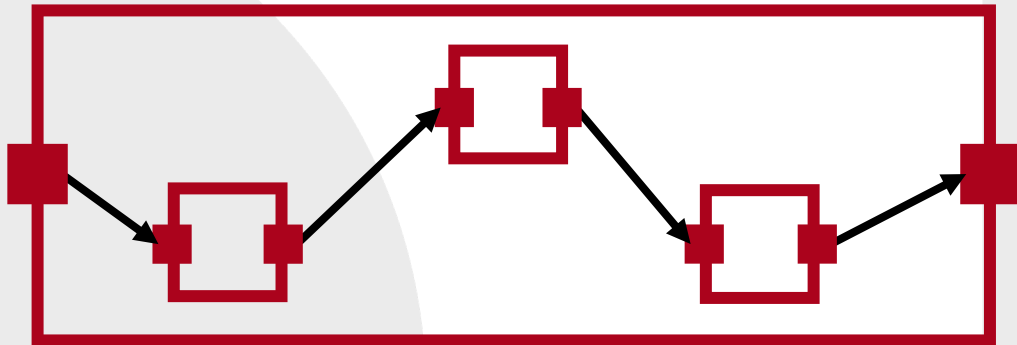
This Lecture - Generate

- **How to come up with a solution?**
- How to refine a solution?
- How to solve a complex design problem?

Which Team Created A **Better** Design?

Team A

Produced **one** detailed design option



Team B

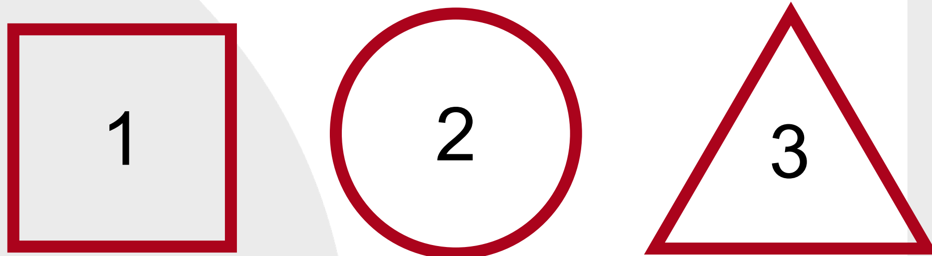
Produced **three** design options

Carnegie Mellon University

Which Team Created A **Better** Design?

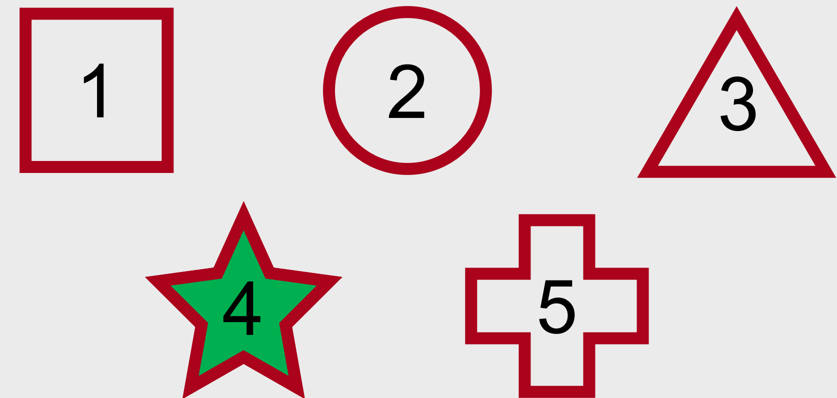
Team B

Produced **three** design options



Team C

Produced **five** design options



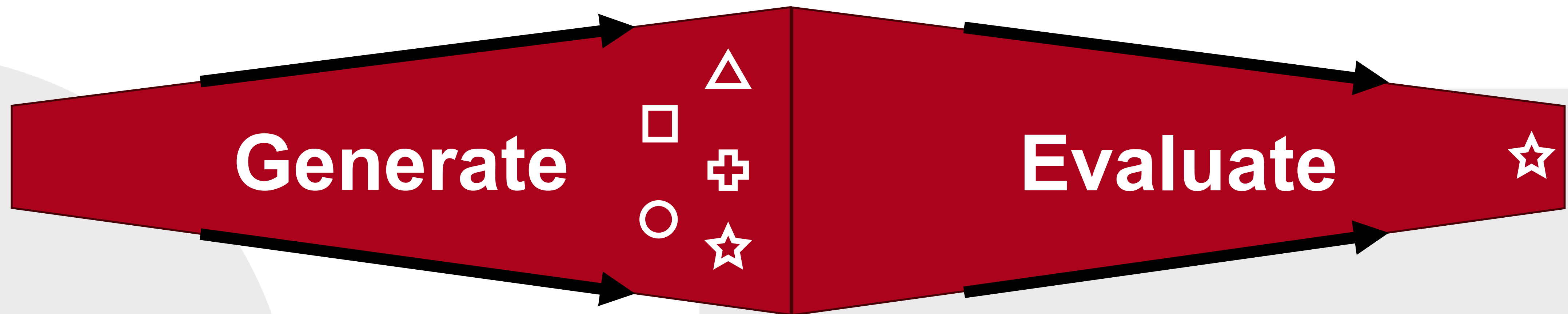
Carnegie Mellon University

Lesson Learned: Think of Many Design Alternatives

- If you can think of a good design, try to **think of a better one**
- Think broadly about a **diverse** range of solution
- Research has shown: When simply prompting designers to **consider other design alternatives**, designers with less experience create **better designs**

Carnegie Mellon University

Lesson Learned:
Think of Many Design Alternatives



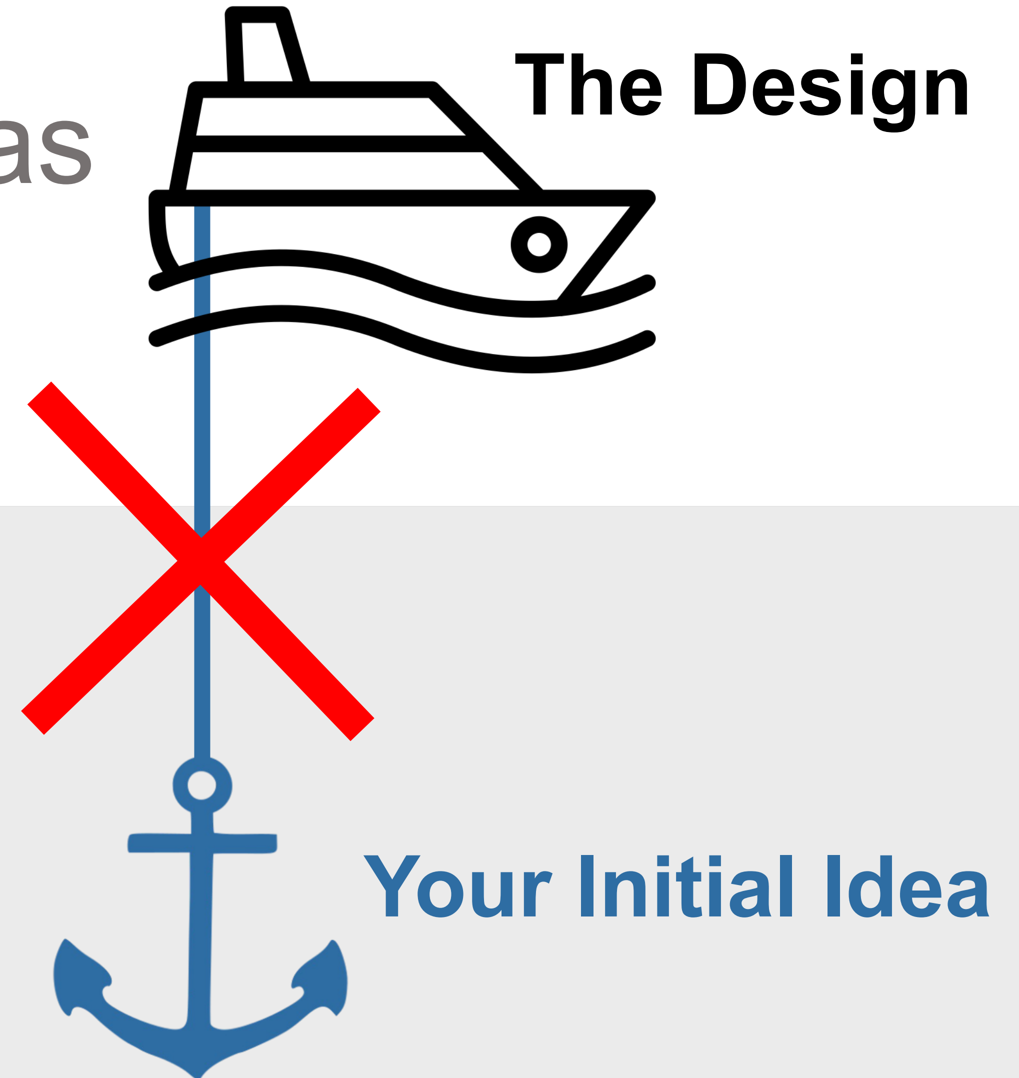
The purpose of idea **generation** is to **broaden up**

Narrowing down of ideas is done in the **evaluation** activity

Carnegie Mellon University

Avoid **Anchoring** to Ideas

- Psychology research shows: People tend to be strongly influenced by their initial ideas and avoid **broader exploration** (i.e., “anchoring”)



Carnegie Mellon University

Lesson Learned: **Avoid Anchoring to Initial Ideas**

- Better: Try to **find ideas that have little in common with your previous ideas**
- **Thinking of weaknesses of your initial ideas** can help to avoid anchoring

Carnegie Mellon University

Example Problem: How to transfer data between different locations

In-Class Activity: Cluster these ideas by similarity

Brainstorming Techniques

1) Write Ideas on Post-Its

RAID is a storage technology for **redundancy** guaranteeing **no data loss** for a certain number of arbitrary disk failures

- Sneaker Net
- Trucks
- Distributed RAID Server
- Decentralized File Sharing
- Central Data Server
- Pigeons

Carnegie Mellon University

Example Problem: How to transfer data between different locations

In-Class Activity: Identify ways to **combine these ideas**

Brainstorming Techniques

2) Cluster Ideas by Similarity

Peer-To-Peer Communication

- Decentralized File Sharing

Client-Server Communication

- Central Data Server
- Distributed RAID Server

Physical Transport

- Trucks
- Pigeons
- Sneaker Net

Carnegie Mellon University

Example Problem: How to transfer data between different locations

Brainstorming Techniques

3) Combine Ideas

Peer-To-Peer Communication

Decentralized File Sharing

Client-Server Communication

Central Data Server

Central RAID Server

Distributed RAID Server

Hybrid Communication

Periodic Local Data Cloning

Carnegie Mellon University

CRC Cards

A common technique for **modelling software design options**

Class / Component / Role [Name of the component]	***Collaborators*** [List of other components that this component starts to interact with]
Responsibilities [Describe this component's obligations to perform a task or know information]	

Carnegie Mellon University

In-Class Activity: Describe relevant **modules**, their **responsibilities**, and **interactions** for this system

Design Exercise: Generate Design Options

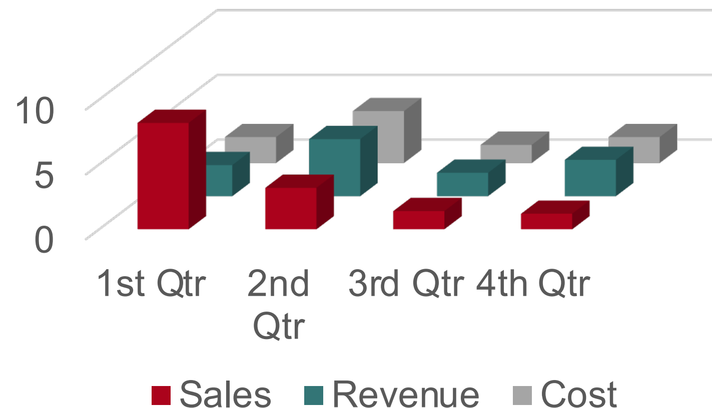
We want to design an **interactive application** that represent the same information across **different views** that should **update immediately** when the information changes.

Table View

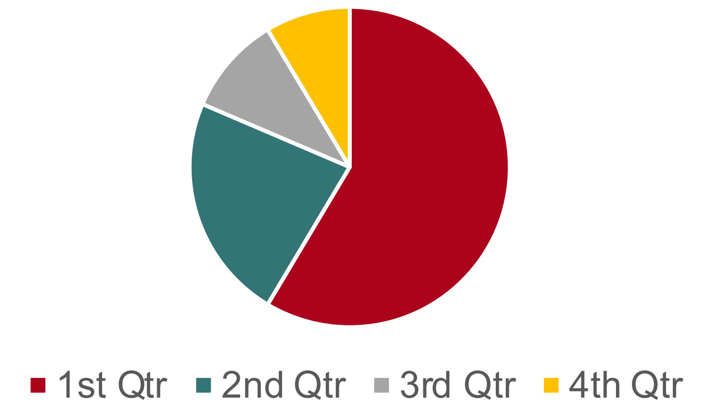
	Sales	Revenue	Cost
1st Qtr	8.2	2.4	2
2nd Qtr	3.2	4.4	4
3rd Qtr	1.4	1.8	1.4
4th Qtr	1.2	2.8	2

Changes here trigger updates in other views

Bar Chart View



Sales Pie Chart View



Carnegie Mellon University

Model-View-Controller (MVC) is a common solution to this problem. MVC is a "**design pattern**".

Model-View-Controller CRC Cards

Component / Role: **Model**	***Collaborators***
Responsibilities - Provides core functionality (main business logic) - Registers views and controllers - Notifies components about data changes	- View - Controller

Component / Role: **View**	***Collaborators***
Responsibilities - Displays information to user - Creates controller - Retrieves data from model - Implements update	- Model - Controller

Component / Role: **Controller**	***Collaborators***
Responsibilities - Accepts user input - Translates evens to service requests for them model or display request to view	- View - Model

Carnegie Mellon University

Lesson Learned: **Start By Considering Existing Solutions**

- Most problems have been **solved already** and described in a **well-documented** way
- **Knowing existing solutions and patterns in your field** can save you a lot of time and effort

Carnegie Mellon University

Design Patterns

It is a **good** solution, but **not always the best** one

found in **many instances**

"A *pattern* is a common, acceptable **solution** to a reoccurring **problem** that arises in a specific **context**"

The problem is generic enough so that the it **generalizes** beyond a few concrete cases

Patterns always refer to a specific **situation**, **goal**, or **trade-off**. Patterns are **not universally** good

Carnegie Mellon University

Pattern descriptions have many **different formats**. At minimum it should describe **problem**, **context**, and **solution**

Example Pattern: *Model-View-Controller*

Context: Designing an interactive application

Problem: How to represent the same information across different views that should update immediately when the information changes?

Solution: Divide your application into model, view, and controller.

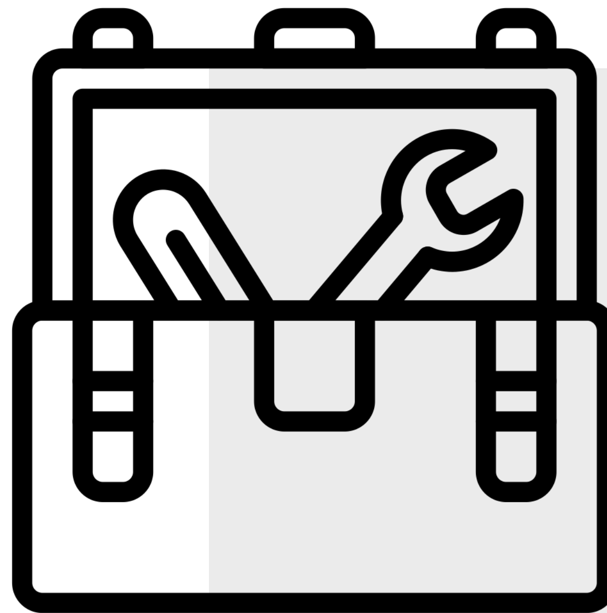
Carnegie Mellon University

Design Patterns are a **Toolbox**

Every **engineer** should know about **common tools for common problems**.

Knowing **advantages** and **disadvantages** of the tools

Knowing when **not to use** a tool



Seeing a problem, **recognizing** which tool is appropriate

Getting **experience** in using the common tools

Carnegie Mellon University

Patterns Describe a **Solution**

- Patterns are **abstractions** of **language-independent** design elements capturing the **core idea** of a solution
  - Involves making design decisions to implementation concretely
  - Patterns abstract away specific project details to **transfer knowledge**

Carnegie Mellon University

Patterns Describe a **Solution**

- Pattern solutions often involve **assigning pattern-specific roles** to classes / objects / messages / components

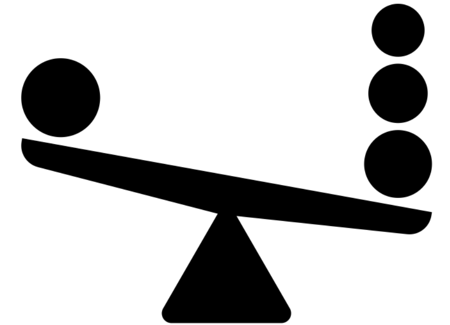
  - Roles superimpose dedicated **responsibilities** on design elements
  - Mentioning roles in naming or documentation **communicates** the design

Carnegie Mellon University

Patterns Describe a **Problem**

- The problem states the **intent & motivation** of the pattern
  - Applied outside of a problem space, a pattern could result in bad design (e.g., overuse of *Singletons*)
- Seeing a pattern in some software tells you not only what the design is, but also **why**

Patterns Describe **Consequences**



- Each design pattern comes with an **inherent trade-off**
  - e.g., design complexity vs. changeability; or performance vs. simplicity
  - Patterns help building software with **well-defined properties**
- Trade-offs need to be evaluated for each **concrete situation**
- Negative consequences can be mitigated by modifying the pattern

Carnegie Mellon University

Question: What **consequences** does MVC have on **quality attributes**?

Consequences of MVC

- **Extensibility of *Views***: Adding new views requires little effort
- **Changeability of *Views***: Changing a view does not require changing other parts of the software
- **Performance of Updates**: Many messages are sent between *models* and *views*
- **Extensibility of *Model***: Adding new features to the model might require changes in *controllers* and *views*.

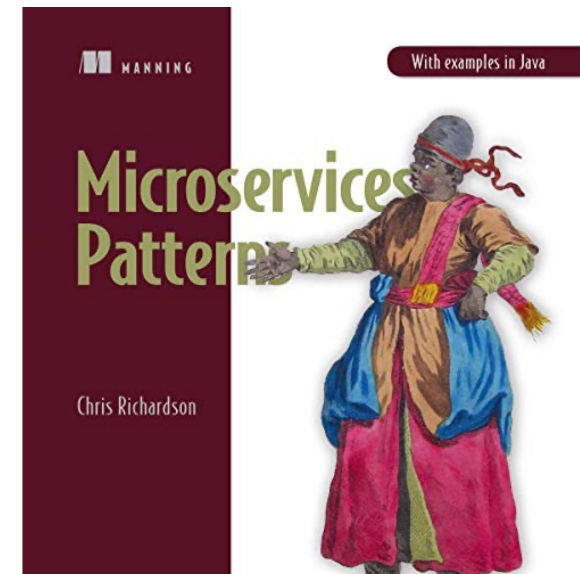
Carnegie Mellon University

Domain-specific Patterns

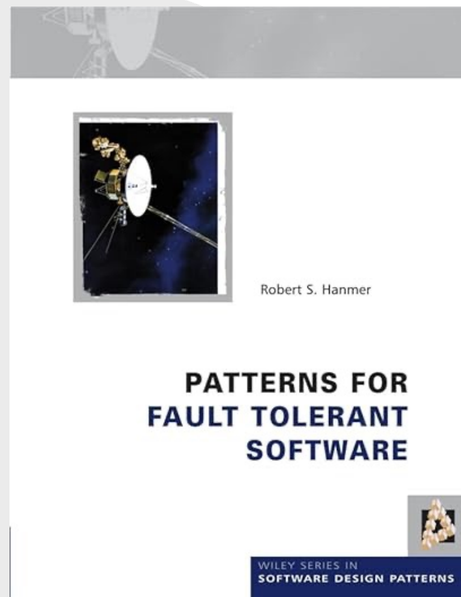
- Each domain has its own patterns and pattern languages
  - Some are variations of generic patterns
- Domain-specific patterns have very high utility in their domain because they can use knowledge about a **sub-context**

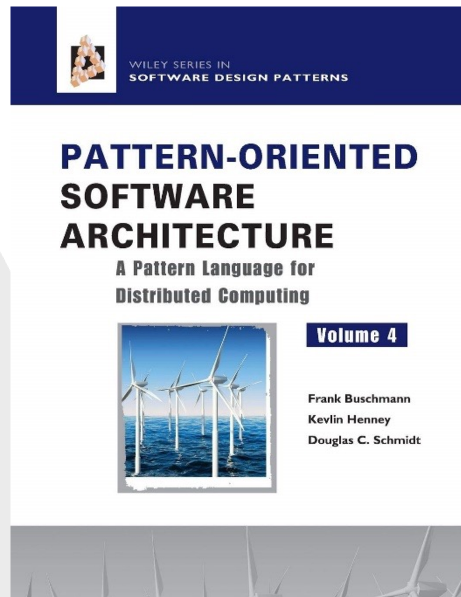
Carnegie Mellon University

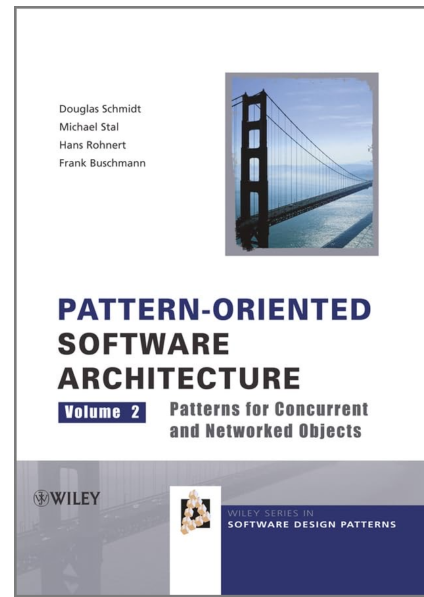
Domain-specific Patterns

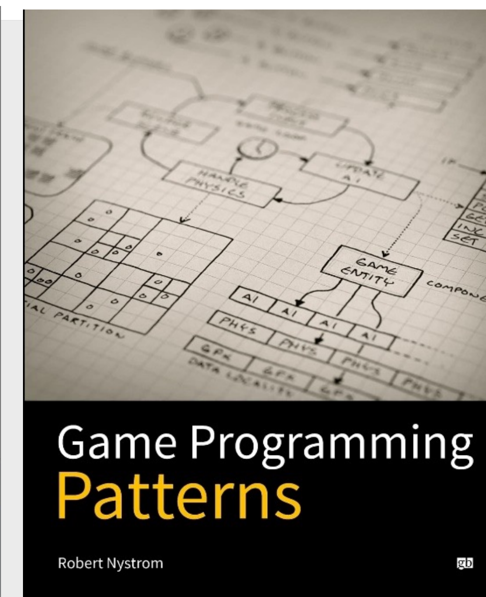


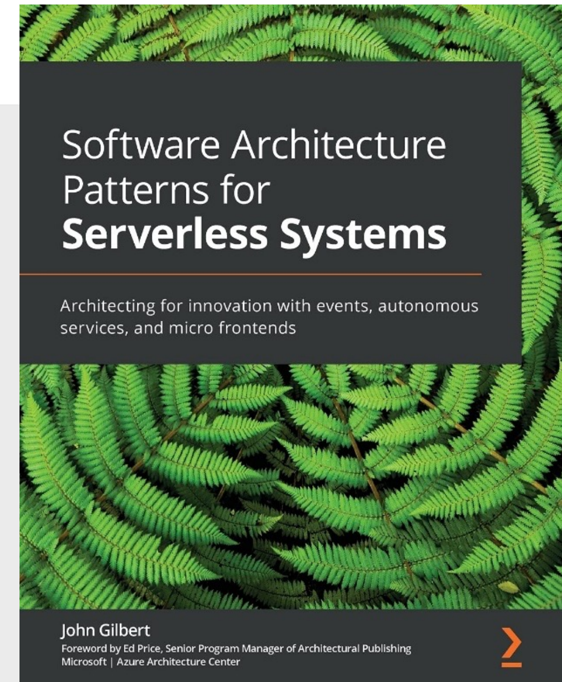
Learn patterns in your domain to become an expert!





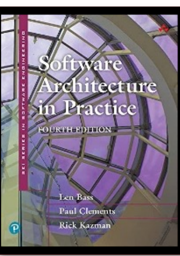






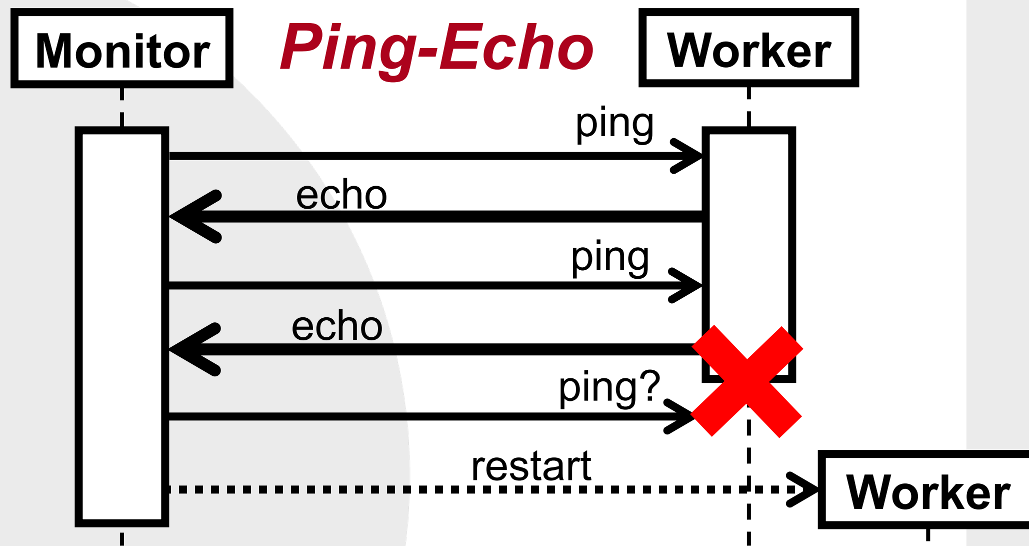
Carnegie Mellon University

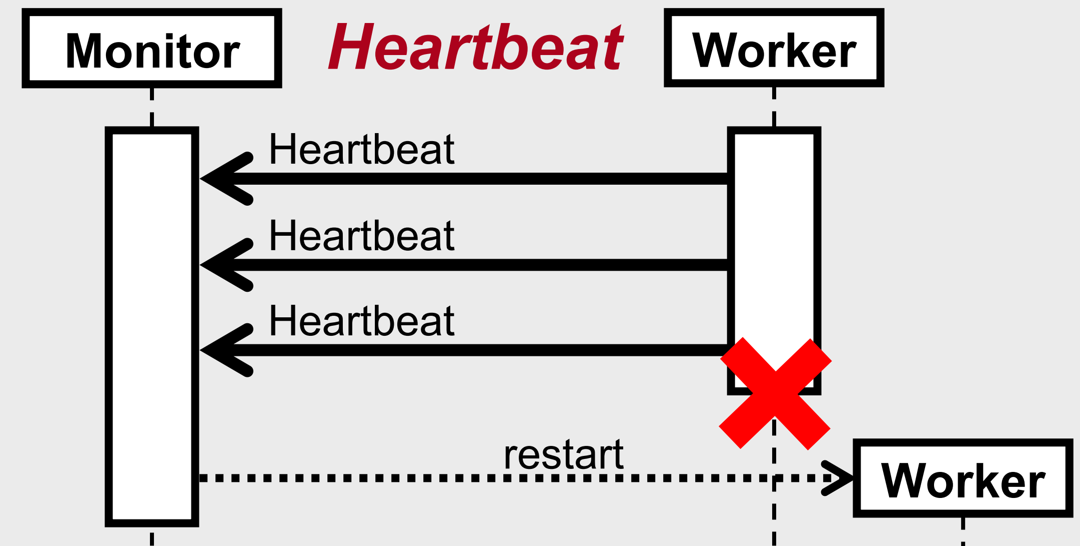
Find more tactics in the book "Software Architecture in Practice"



Other Types of Design Reuse: **Tactics**

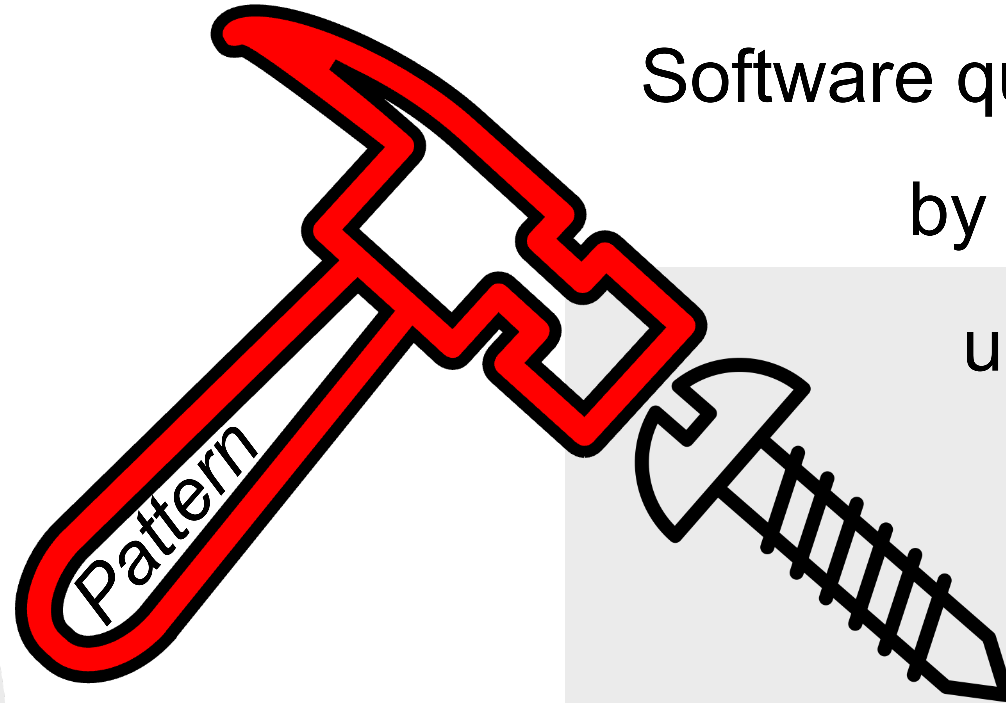
- Tactics describe common ways to improve a quality attribute
- E.g., ***Ping-Echo*** and ***Heartbeat*** improve **Availability**





Carnegie Mellon University

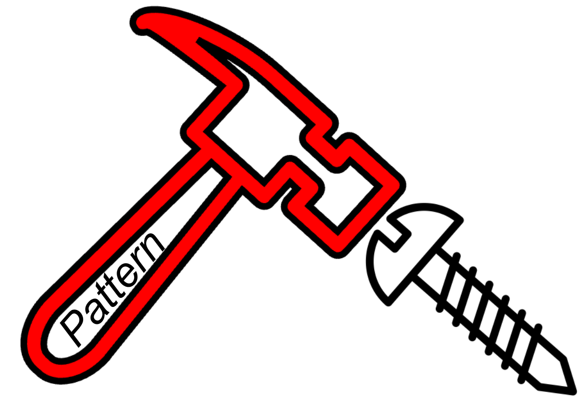
If you have a **hammer**, everything looks like a **nail**



Software quality is **not** measured by the number of used patterns

Carnegie Mellon University

Lesson Learned: **Avoid Over-Using Design Patterns**



- Design patterns are a common source of **anchoring**

- Consider **context** & **consequences** thoroughly before choosing a pattern
- Think of **many alternatives** to design patterns

Other Common Challenges with Patterns

- Patterns can be **misinterpreted as recipes**
  - Integration of patterns is a human-intensive, manual activity
  - Patterns don’t make domain expertise obsolete
- When applying them, it is important to **tailor** them to the concrete context

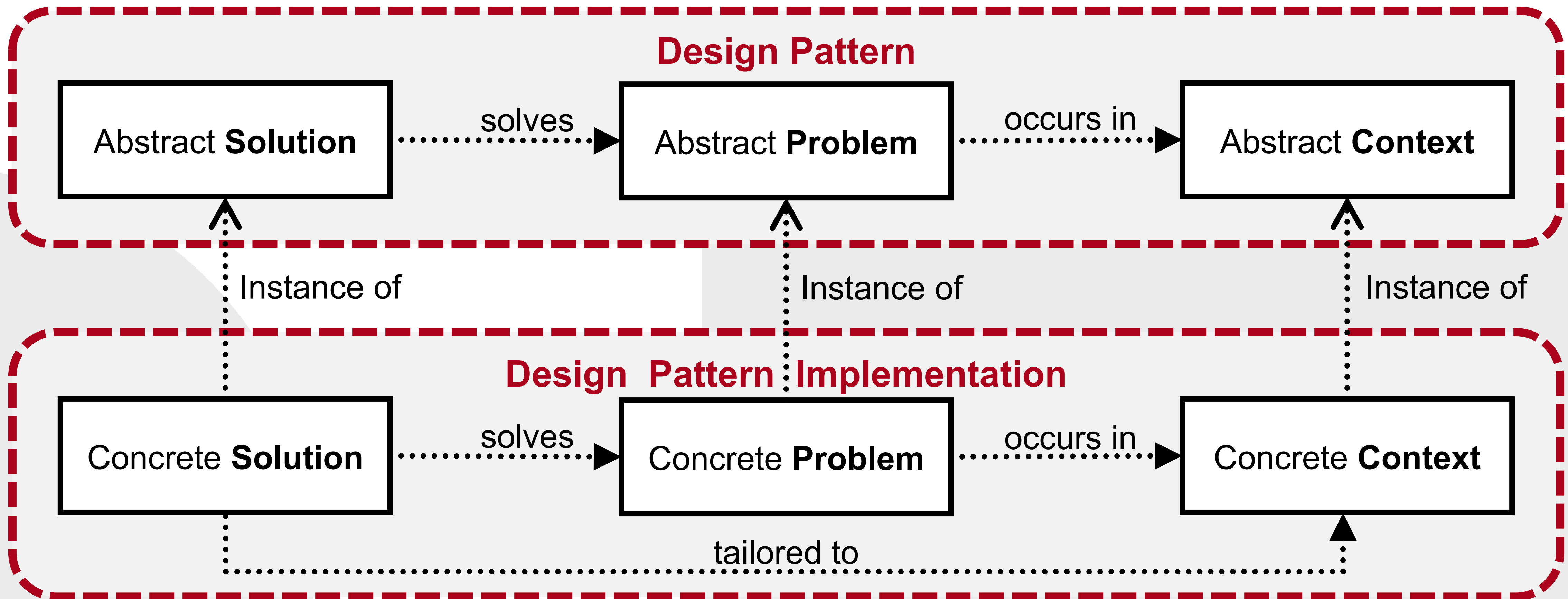
Carnegie Mellon University

This Lecture - Generate

- How to come up with a solution?
- **How to refine a solution?**
- How to solve a complex design problem?

Carnegie Mellon University

Design Pattern Instantiation



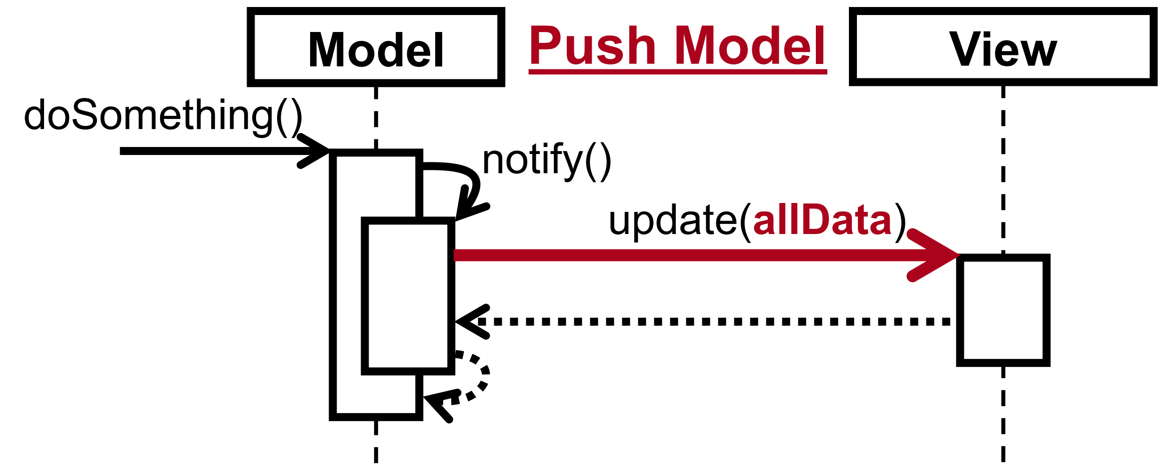
Variation Points

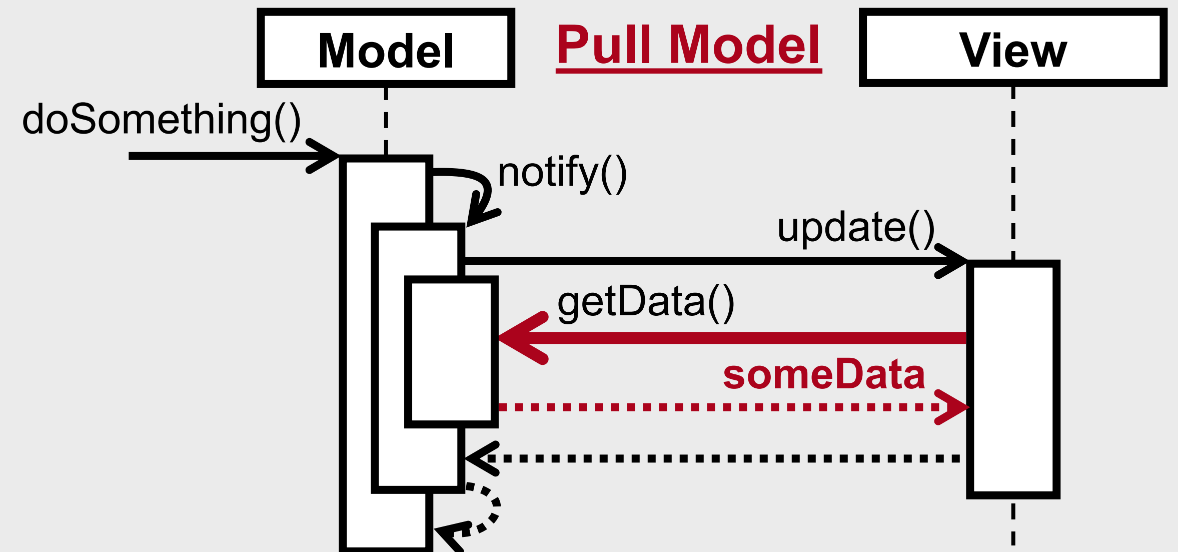
- Variation points are **unresolved design decisions** of a reusable design
- Good pattern descriptions explicitly document variation points
- When instantiating a pattern, **think of possible variation points**

Carnegie Mellon University

MVC Variations

- Push model: *model* pushes **all updated data**
- Pull model: after updates, *views* and *controllers* pulls data on a **need-to-know basis**



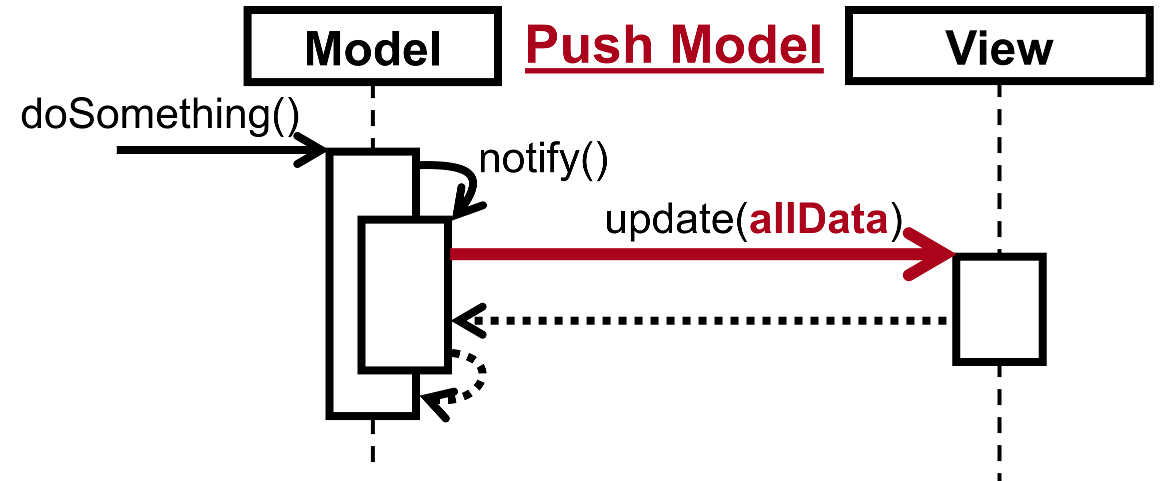


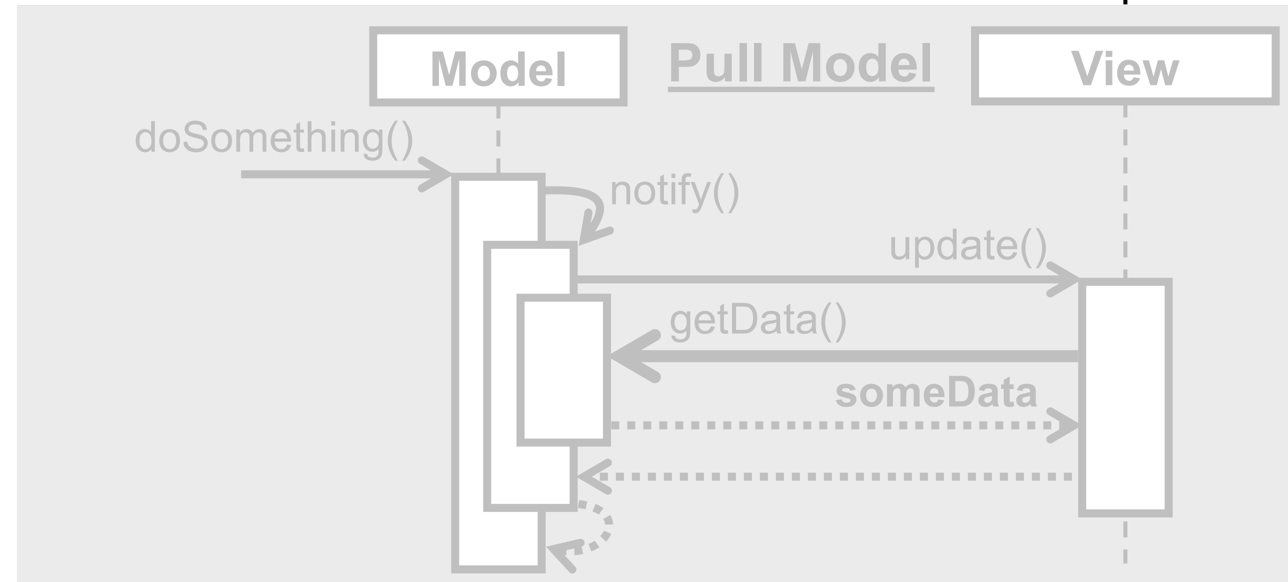
Carnegie Mellon University

Question: What **consequences** does the Push Model have?

Push Model

- Model pushes **all updated data** to controllers and views
- Fewer **calls / messages**
- **Simpler** API
- Potentially huge data structure is passed, increasing **coupling**

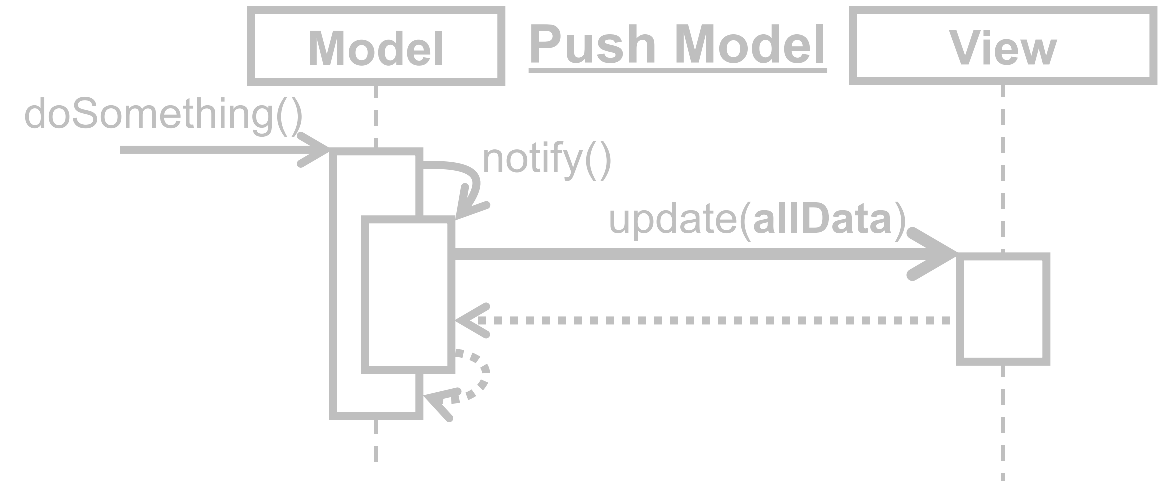


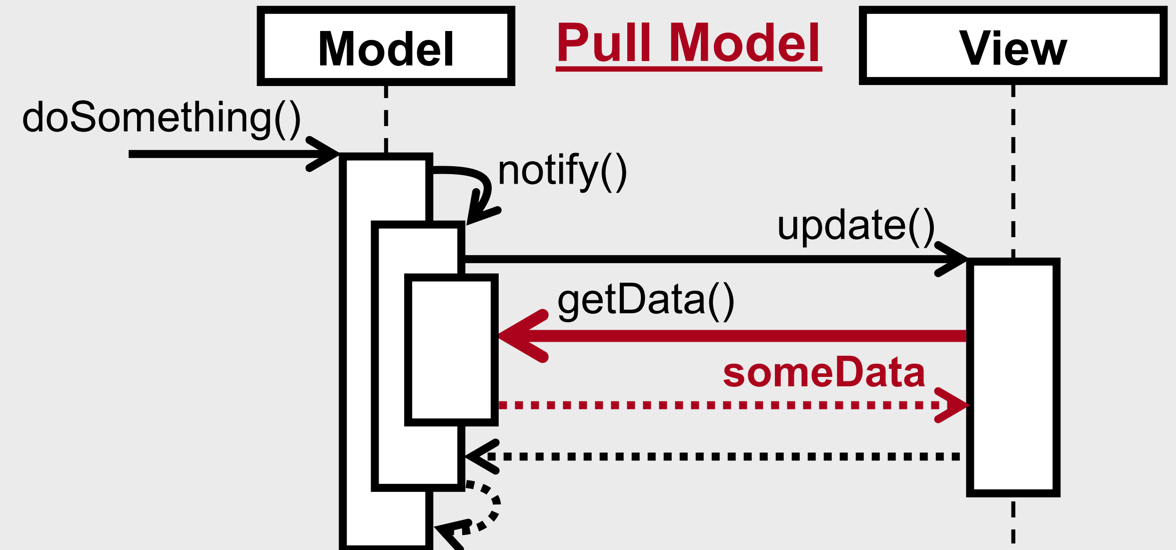


Question: What **consequences** does the Pull Model have?

Pull Model

- *Views* and *controllers* pulls data on a **need-to-know basis**
- *Model* **does not need to know** what *views* are displaying data in
- On distribuend networks: more options for **confidentiality**
- Potentially **inefficient**





Carnegie Mellon University

Lesson Learned: **Consider Variation Points When Tailoring a Pattern To Your Context**

- Design patterns need to be **tailored** to the **concrete context**
- Each design pattern describes a broad **design space** with many **variations**
- Variations can impact the **consequences** of a design pattern

Carnegie Mellon University

Lesson Learned: **Refine A Design By Identifying And Resolving Variation Points**

Variation points at different levels of abstraction form a **decision tree**

client server vs. **peer to peer?**

How to prevent data loss?

How to discover peers?

RAID

Backups

Which RAID level?

How often to backup?

Carnegie Mellon University

This Lecture - Generate

- How to come up with a solution?
- How to refine a solution?
- **How to solve a complex design problem?**

Carnegie Mellon University

There is **more than one** design problem to solve. So, we cannot immediately start generating ideas after understanding requirements

What Is **Different** About Generating Ideas for **Complex System**?

17-423/723 Course Project

The goal of this project is to help you gain hands-on experience applying design principles and techniques from this class by designing, implementing, and iteratively improving a complex software system. In particular, you will work in teams to design, implement, and deploy a scheduling system for medical appointments, like those used by the public to schedule testing and vaccine appointments during a pandemic.

Although scheduling might seem like an easy task, it has multiple layers of complexity that makes it a challenging design problem. There are a number of different stakeholders (e.g., users/patients, pharmacies, hospitals, medical personnel, test/vaccine suppliers, policy makers) with competing requirements and constraints. The requirements may evolve as new types of testing/vaccine requirements arise or medical supplies fluctuate over time. During the pandemic, within the US, there were ambitious plans for building a unified, nation-wide or state-wide scheduling app, many of which ended up being far less than successful (ex1, ex2, ex3). These

Carnegie Mellon University

Difference to Refinement and Variation Points: Here the sub-problems are not part of the solution we picked but inherently part of the main problem

Lesson Learned: **Divide And Conquer To Solve Complex Problems**

- Split a complex problem into smaller sub-problems
- Solve sub-problem first, then combine them
- Do not forget about the whole
- Reflect about the relationships of the parts
- After merging sub-solutions, adjust the if necessary

Carnegie Mellon University

Delaying Decisions

- Identify design decisions that need more information or that are likely to change later
- Attempt to design your system without assuming a solution for these difficult decisions
- Keep a list of delayed decisions and keep track of what you need to resolve them

See **Information Hiding Principle** (next lecture)

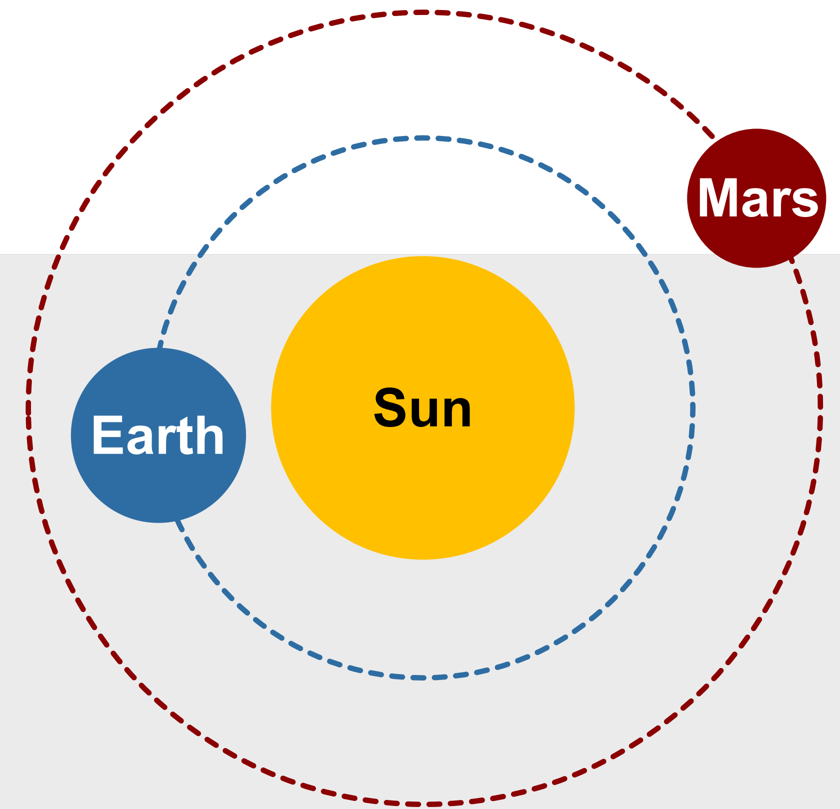
Carnegie Mellon University

Problems that arise when messaging **on Earth**, added infrastructure, networking with largely varied distances, and different definition of a day

Design Exercise: Divide And Conquer

Design an interplanetary messaging system for people living on **Earth** and **Mars** to communicate with each other!

What sub-problems would you need to solve?



Carnegie Mellon University

Lesson Learned: **Solve Simpler Problems First**

- When faced with a complex problem, experts solve a simpler problem first
- Solution to simpler problem might be incomplete but can be **extended later**
- Be aware when the simpler problem is so fundamentally different that solutions do not generalize

Carnegie Mellon University

Please Complete the Exit Ticket in Canvas!

Question 1 1 pts

Please list **three tips** from this lecture that you could give a friend on how to generate design alternatives more effectively.

Question 2 1 pts

Please describe one common **advantage** and one common **challenge** of using design patterns in software design.

Question 3 1 pts

Please leave any questions that you have about today's materials and things that are still unclear or confusing to you (if none, simply write N/A).

Carnegie Mellon University

Summary

- Think of Many Design Alternatives
- Avoid Anchoring to Ideas
- Start By Considering Existing Solutions
- Avoid Over-Using Design Patterns
- Divide And Conquer to Solve Complex Problems
- Solve Simpler Problems First