Week 6 - Wednesday

COMP 3100

Last time

- What did we talk about last time?
- User interaction design
- Swing

Questions?

Design Principles

SAC principles

- **Design principles** favor certain characteristics to make a design better
- SAC principles are three general interaction design principles
- **Simplicity**
  - Simple designs are better
  - Lots of options are confusing for the user
  - It's better to make commonly used options easy and require a little more work for unusual options
- **Accessibility**
  - Designs that can be used be more people are better
  - Considerations: color blindness, things too small to see or interact with
- **Consistency**
  - Designs that present data in similar ways are better
  - Example: use consistent navigation controls

CAP principles

- CAP principles are focused on appearance
- **Contrast**
  - Designs that make different things obviously different are better
  - Example: italics and bold
  - Example: font size to distinguish headings from text
- **Alignment**
  - Designs that line up on a grid are better
  - Indentation is useful
- **Proximity**
  - Designs that group related things together are better

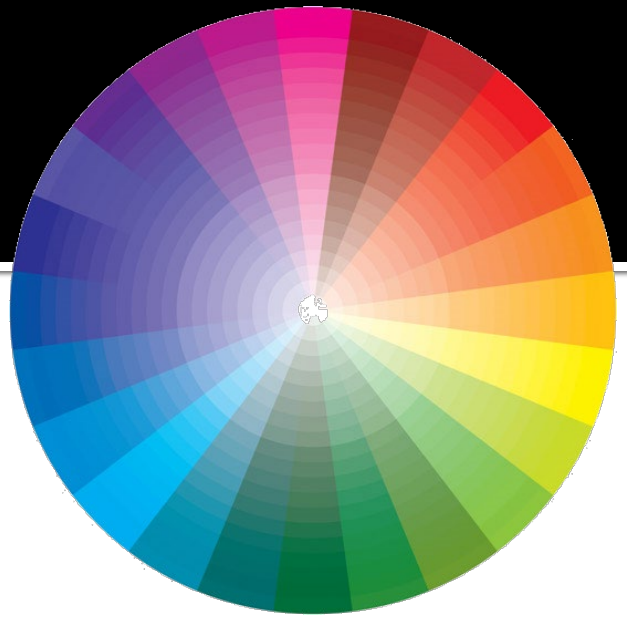
FeVER principles

- FeVER principles are about behavior
- **Feedback**
  - Designs that acknowledge user actions are better
  - Otherwise, how do you know if what you're doing has an effect?
- **Visibility**
  - Designs that display their state and available operations are better
  - Are we in Arm Bomb or Disarm Bomb mode?
- **Error Prevention and Recovery**
  - Designs that prevent user errors and allow error recovery are better
  - Prevention example: disable buttons that shouldn't be pressed
  - Recovery example: allow undo or ask "Are you sure?" before doing something dangerous

Visual Design Tips

Color

Color terminology



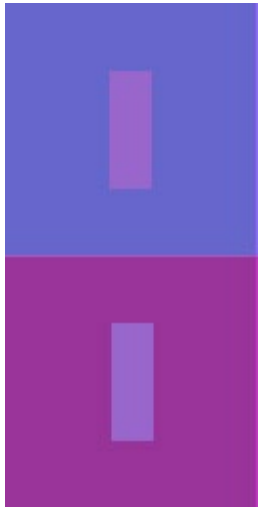
- Artists divide colors into
  - **Primary**: **red**, **blue**, **yellow**
  - **Secondary:** mixes of primaries, **orange**, **green**, and **purple**
  - **Tertiary**: combinations of primaries and secondaries
- **Complementary colors** are across from each other on the color wheel, with strong contrast
- **Analogous colors** are next to each other on the color wheel

Colors and psychological impact

- **Warm colors** like red, yellow, and orange are on one side of the fence
  - Red has associations with strong archetypes, like hatred, passion, and fire
  - Yellow is tied to happiness and sunshine
- **Cool colors** like green, blue, and purple are on the other
  - Blue is cold and serene and trustworthy
  - Green is associated with nature, the environment, healing, and money
  - Purple is connected to royalty and femininity
- **Neutral colors** like gray and brown do not evoke much emotion
  - Black can run the range from authority to death to mystery to elegance
  - White can show purity, innocence, cleanliness, winter
- Colors evoke feelings in people, but there is so much variation

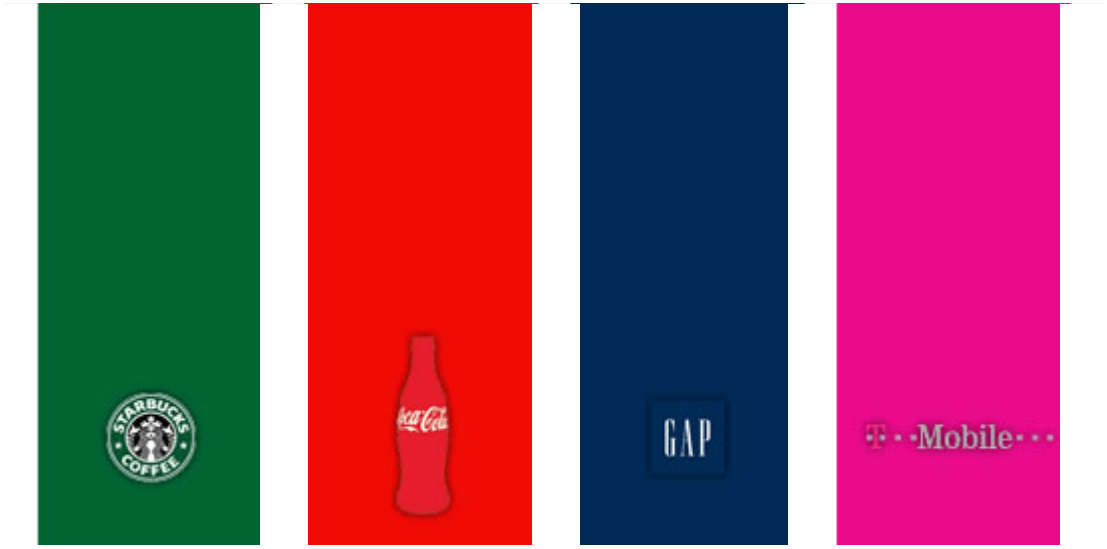
Backgrounds

- A white background is the standard for readability
- Black backgrounds are often used to give a more exciting, youth-oriented, or "cool" feel
- Background colors are important since a single color can look different on different backgrounds

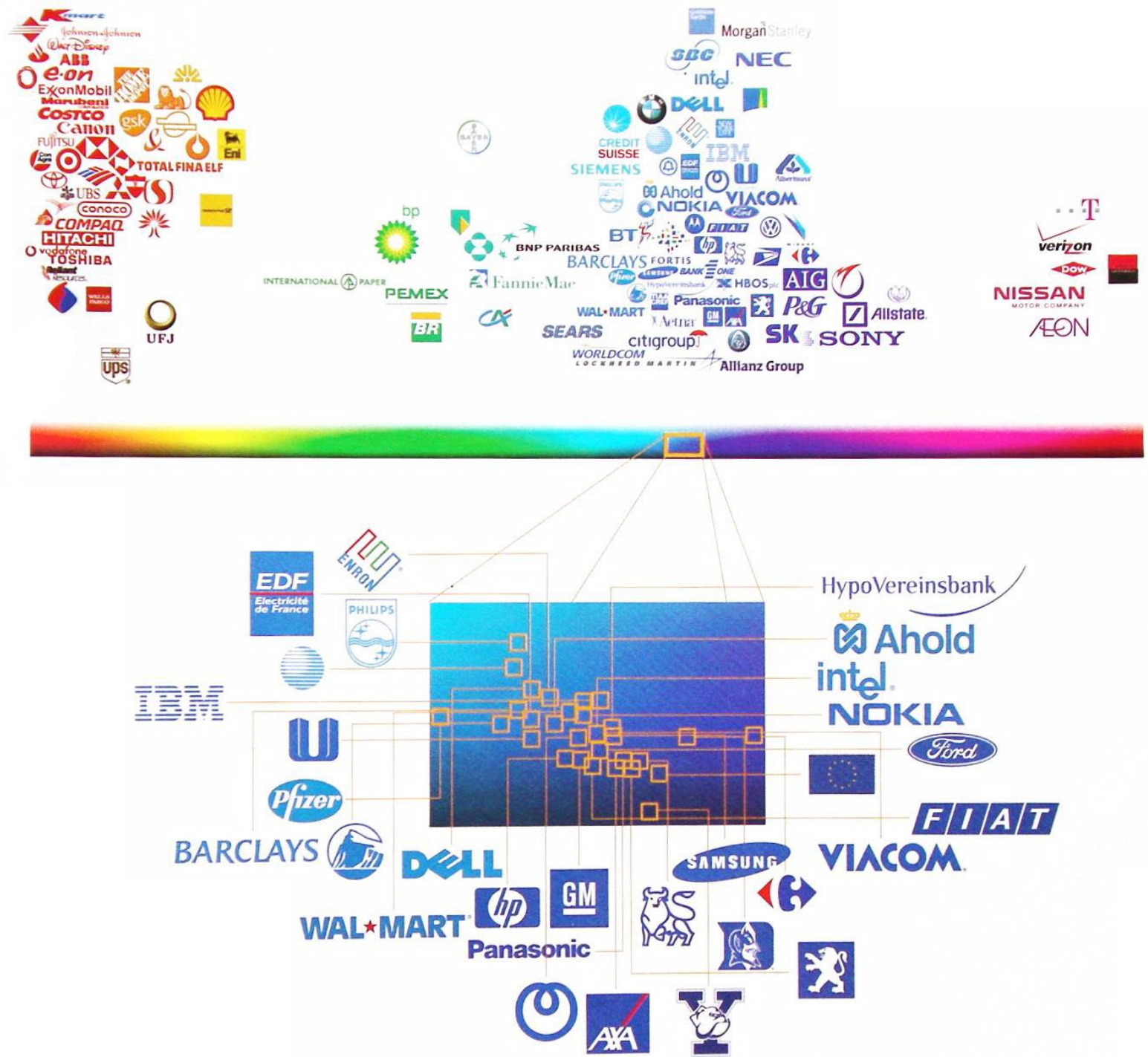


Identity

- Corporate branding is heavily tied to recognizable colors



- T-Mobile has even sued other companies for using magenta (unsuccessfully, it seems)



Graphic from Wired magazine 2003

Color tools

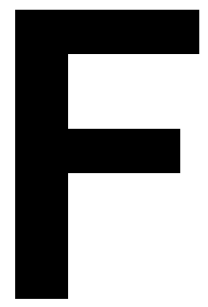
- Google Lighthouse can check a website for contrast and other issues
  - It's a tool built into Chrome
- Toptal is a website that lets you see what webpages look like for people with certain kinds of color blindness
  - https://www.toptal.com/designers/colorfilter

Typefaces

Two major classes of fonts

- **Serif** fonts have little pointy bits

- **Sans serif** fonts do not



Picking a font

- Make sure it is readable wherever it will display (books might be different from on screen)
- Read your content out loud in the font you pick to test readability
- Consider what audience you're trying to appeal to
- What is the setting where your text will appear?
- The process is subjective; there may not be a solution that pleases everyone

Typography for readability

- Sans serif is usually considered more readable on screens
- Avoid ALL CAPS
- For headings, **down style** (capitalizing only the first word and proper nouns) is more readable than **up style** (capitalizing every major word)

Pairing fonts

- Using lots of fonts is amateurish and looks like a ransom note
- Using a single font is reasonable, especially for a webpage
- Some designers prefer to **pair fonts**, using two distinct fonts, often one for headings and the other for text
  - An alternative is to use two very different weights of the same font
- The fonts should be different enough that they are not confused
  - Ideally from different families
- Serif for headings and sans serif for body is a good stand by
- Sometimes the headings will be images of text rather than text

Other font guidelines

- Contrast is key
  - Easy with black text on white background or vice versa
  - If you are using colored fonts or backgrounds, how much contrast is there if you were to change to grayscale?
- Size
  - Don't go too small!
  - 10pt or 12px is the smallest your fonts should be
- Hierarchy
  - Use different fonts or weights or styles for title, heading, and body text
  - Be consistent!
- Negative space is important
  - Lets the eye rest
  - Keeps the content from running together

More design principles

- Alignment and proximity
- Balance
- Consistency and repetition
- Contrast and whitespace
- Gestalt
- Golden ratios

Alignment and proximity

- All the images and text in your layout should follow some plan for alignment
  - Horizontal
  - Vertical
  - Edges aligned
  - Everything centered
  - "Visual" alignment
- Unaligned items usually look sloppy
- It's possible to break alignment for a dramatic effect
- Items being close together suggests that they relate to each other

Balance

- Evenly distribute text and graphics
- Symmetrical balance divides things down the middle
- Asymmetrical balance leans to one side to create emphasis or interest
  - Visual center is not always the center of the page
- In radial balance, elements radiate from the middle
- The rule of thirds suggests that a design can be made interesting by dividing the space into thirds such that important elements fall into only one of the three

Symmetrical

Asymmetrical

Radial

Consistency and repetition

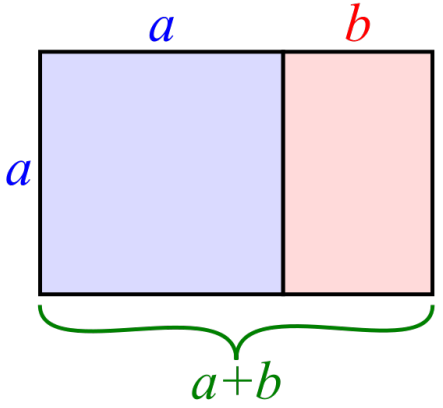
- Repetition is often a good thing in design
- Use the same styles and fonts consistently for top level headings, body text, quotes, etc.
- Use consistent layout for navigation bars and layout
- Page numbers or copyright notices should be in the same place on each page
- Repetition is comforting to readers
  - They will often complain if there is any change, even if the new design is an improvement

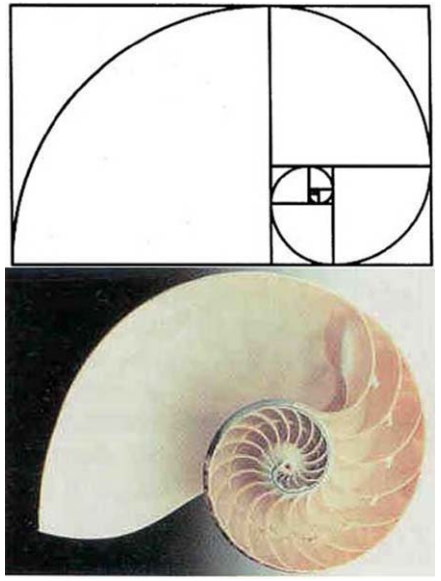
Gestalt

- Gestalt is the idea that the whole is different than the sum of its parts
- Each individual part of a design can seem good on its own, but it might not fit together right
- Think about figure and ground
- Things that are incomplete are perceived as whole
- Many of the guidelines about grouping similar items together or making things with similar functions look similar come from gestalt psychology

Golden ratio

- The golden ratio (or mean) is φ (phi), approximately 1.6180339887498 9 and even more approximately 5:3
  - This value occurs in the closed-form solution of Fibonacci
- This ratio occurs in nature and is considered to look pleasing in art and design
- It comes from the relationship $a + b : a$ as $a : b$
- This ratio is a good guideline for dividing up columns or organizing other layout





Software Engineering Design

Software engineering design

- **Software engineering design** is designing programs, sub-systems, and their constituent parts
- Product design and interface design specifies the external features of a product
- Engineering design specifies the internal features
  - Making it work
- When designing, it can be useful to make many different models showing different views of the system
  - Class diagrams breaking the system into its parts
  - State diagrams showing the states it can be in
  - Diagrams showing hardware and software interactions

Preventing design defects

- A number of approaches can be used to avoid design defects
- **Design principles:** Using a list of good principles helps you make good choices
- **Design notations:** Using good notations (often UML diagrams) helps designs be complete and consistent
- **Design processes:** Using established processes for designs helps avoid mistakes
- **Design patterns:** Using patterns, models designed to be imitated, reuses solutions that have worked in the past (and make design easier)

Active review

- The previous slide was about *preventing* design defects
- To *detect and remove* design defects, an effective technique is **active review**
- In an active review, experts answer questions about parts of the design
- Active reviews have three phases:
  - **Preparation:** Designers choose parts of the design they aren't happy with, choose experts to examine each part, and prepare questions for the experts
  - **Performance:** Reviewers get the design and answer the questions
  - **Complete:** Designers read the responses and update their design

Design principles

- As with interface design, there are many software **design principles** that can help us evaluate the quality of a design
- Some important design principles are:
  - **Simplicity**
  - **Small modules**
  - **Information hiding**
  - **Minimize module coupling**
  - **Maximize module cohesion**
- In this context, **module** means any meaningful program unit
  - For OOP, classes or methods are usually considered modules
  - The definition is intentionally vague to cover many different languages
  - It does **not** mean Java modules (an organization level above packages in Java 9+)

Simplicity

- As with interface design, simpler designs are better
- What is simplicity?
  - Fewer lines of code
  - Fewer control structures
  - Fewer connections between different parts
  - Fewer computations with different kinds of objects
- A good rule of thumb is which design is easiest to understand
- Simplicity is a good goal, but some important algorithms in computer science are necessarily complex

Everything should be made as simple as possible, but not simpler.

-Attributed to Albert Einstein, who probably did not say it quite like that

Small modules

- Designs with small modules are better
- Smaller modules are easier to read, to write, to understand, and to test
- Miscellaneous guidelines:
  - Classes should have no more than a dozen operations (methods)
  - Classes should be no more than 500 lines long
  - Operations should be no more than 50 lines long
  - I have heard that you should be able to cover a method with your hand
- Of course, it is often impossible to follow these guidelines

Information hiding

- Each module should shield the internal details of its operation from other modules
- Declare variables with the smallest scope possible
- Use **`private`** (and **`protected`**) keywords in OOP languages to hide data (and even methods) from outside classes
- Advantages of information hiding:
  - Modules that hide their internals can change them without affecting other things
  - Modules that hide information are easier to understand, test, and reuse because they stand on their own
  - Modules that hide information are more secure and less likely to be affected by outside errors
- This is why we use mutators and accessors instead of making members public

Minimize module coupling

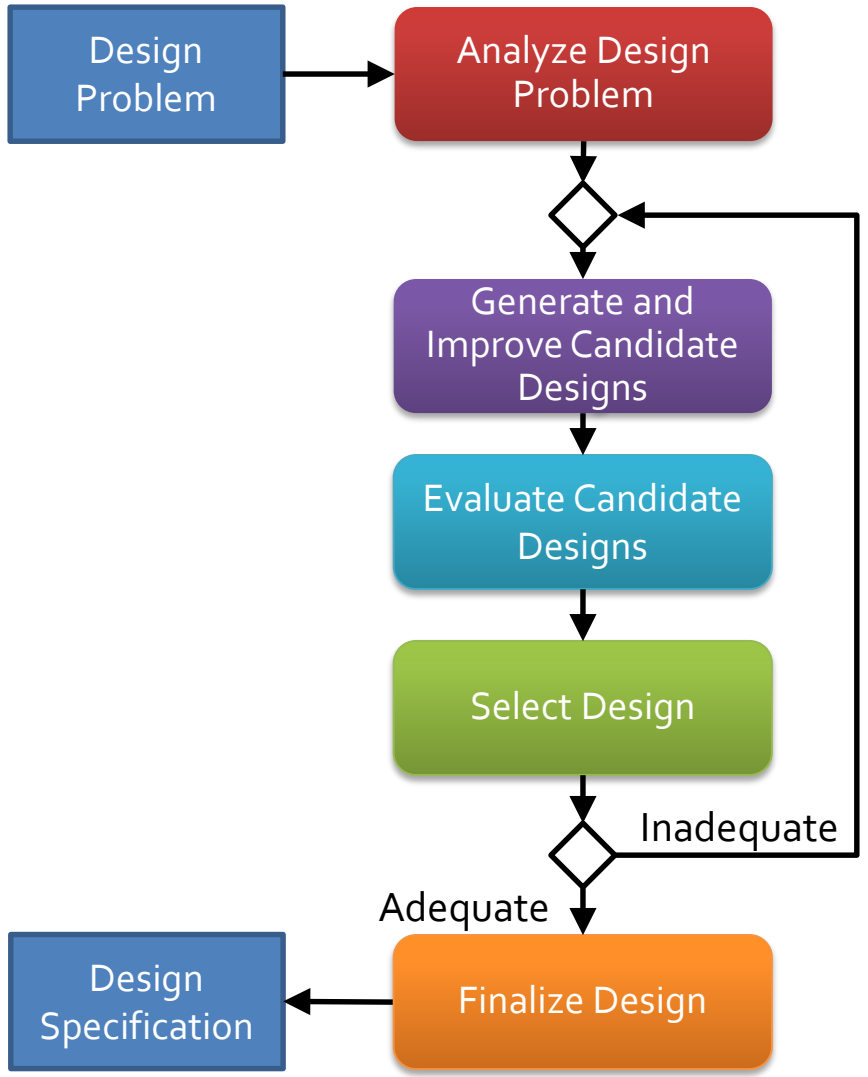
- **Module coupling** is the amount of connectivity between two modules
- Modules can be coupled in the following ways:
  - One class is an ancestor of another class
  - One class has a member whose type is another class
  - One class has an operation (method) parameter whose type is another class
  - One operation calls an operation on another class
- If there two modules have many of these couplings, we say that they are **strongly coupled** or **tightly coupled**
- When modules are strongly coupled, it's hard to use them independently and hard to change one without causing problems in the other
- Try to write classes to be as general as possible instead of tied to a specific problem or set of classes
- Using interfaces helps

Maximize module cohesion

- **Module cohesion** is how much the stuff in the module is related to the other stuff in the module
- We want everything in a class to be closely related
- It's best if a class keeps the smallest amount of information possible about other classes
- More module cohesion usually leads to looser module coupling
- Sometimes a module being hard to name suggests that its data or operations are not cohesive

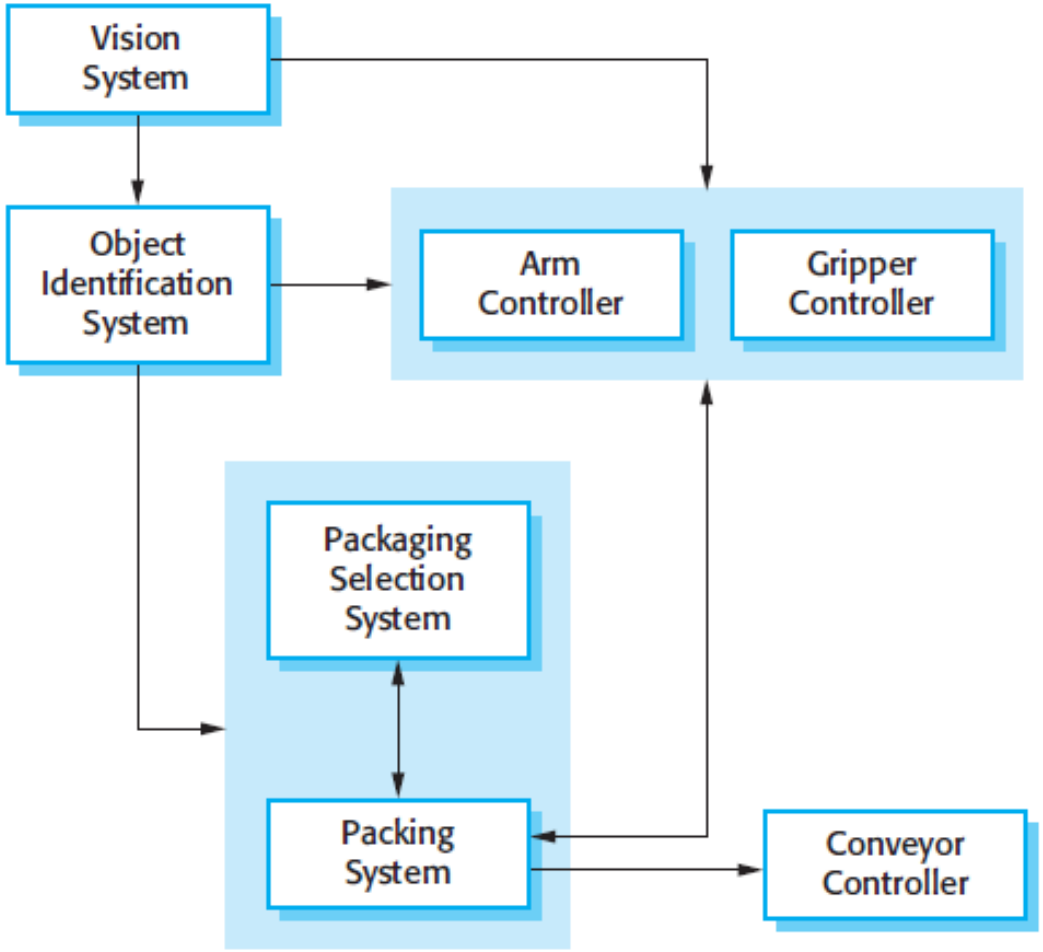
Design process

- The design process is a microcosm of the larger software development process
- The steps are analyzing the problem, proposing solutions (and looking up existing solutions to similar problems), and evaluating the solutions (perhaps combining different solutions) until a design is selected



Architectural design

- **Architectural design** is specifying a program's major components
- Architectural design is often modeled with a **box-and-line diagram** (also called a block diagram)
  - Components are boxes
  - Relationships or interactions between them are lines
  - Unlike UML diagrams, box-and-line diagrams have no standards
  - Draw them in a way that communicates your design

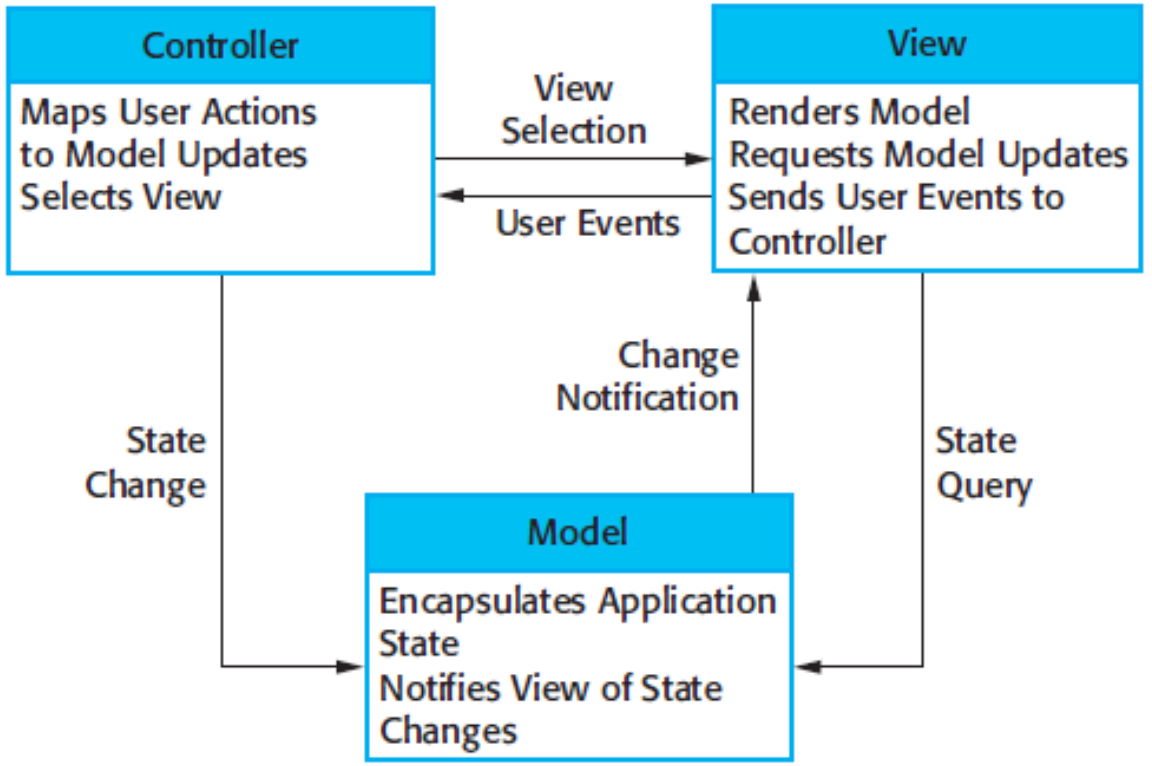


Architectural styles

- **Architectural styles** are patterns that can be followed for architectures
  - Model-view-controller
  - Layered architecture
  - Repository architecture
  - Client-server architecture
  - Pipe and filter architecture
- Architectural styles are the high-level analog of design patterns
- These styles can be used for parts of your design or combined into hybrid styles

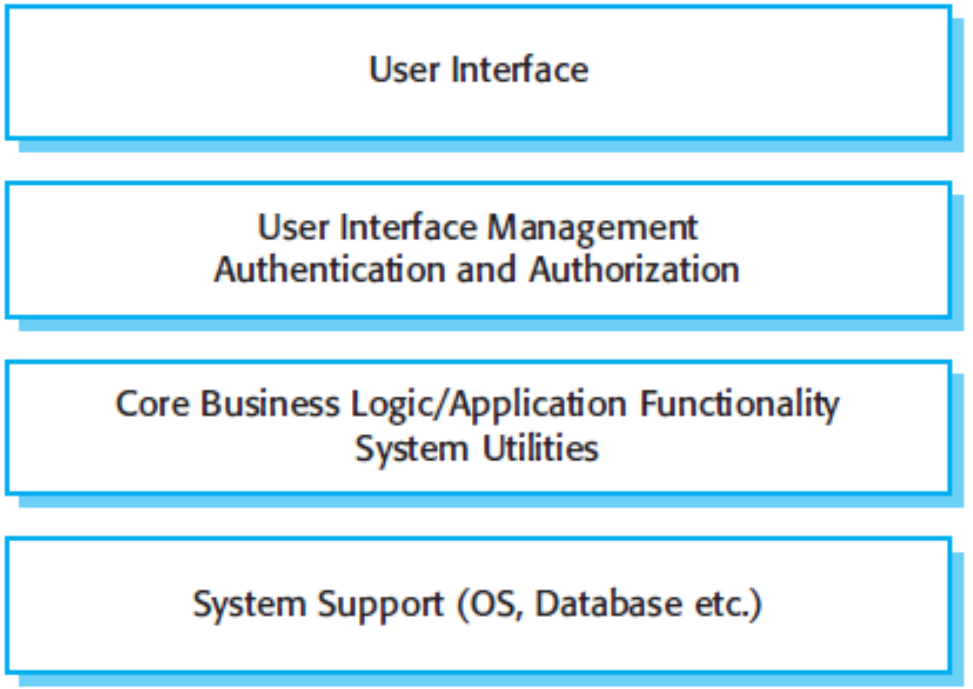
Model-View-Controller

- The **Model-View-Controller** (MVC) style fits many kinds of web or GUI interactions
- The **model** contains the data that is being represented, often in a database
- The **view** is how the data is displayed
- The **controller** is code that updates the model and selects which view to use
- The Java Swing GUI system is built around MVC
- Good: greater independence between data and how it's represented
- Bad: additional complexity for simple models



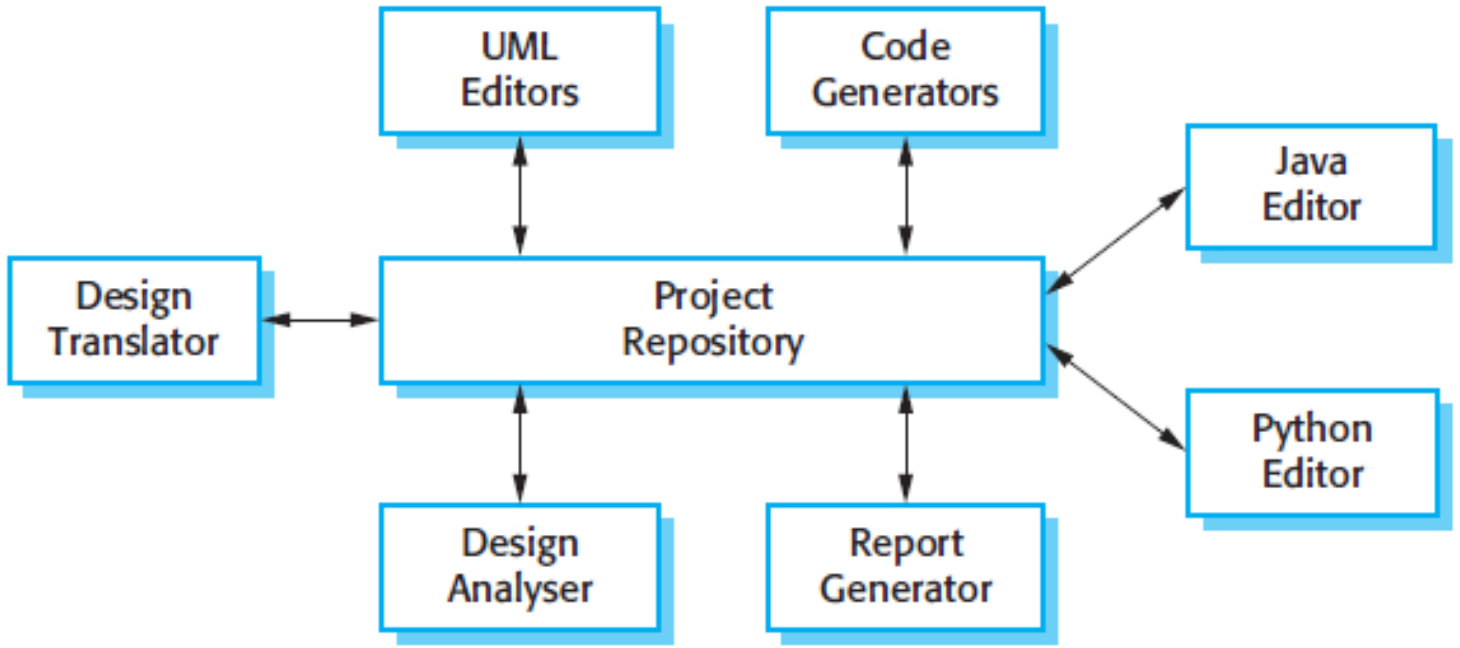
Layered style

- Organize the system into layers
- Each layer provides services to layers above it, with the lowest layer being the most fundamental operations
- Layered styles work well when adding functionality on top of existing systems
- Good: entire layers can be replaced as long as the interfaces are the same
- Bad: it's hard to cleanly separate layers, and performance sometimes suffers



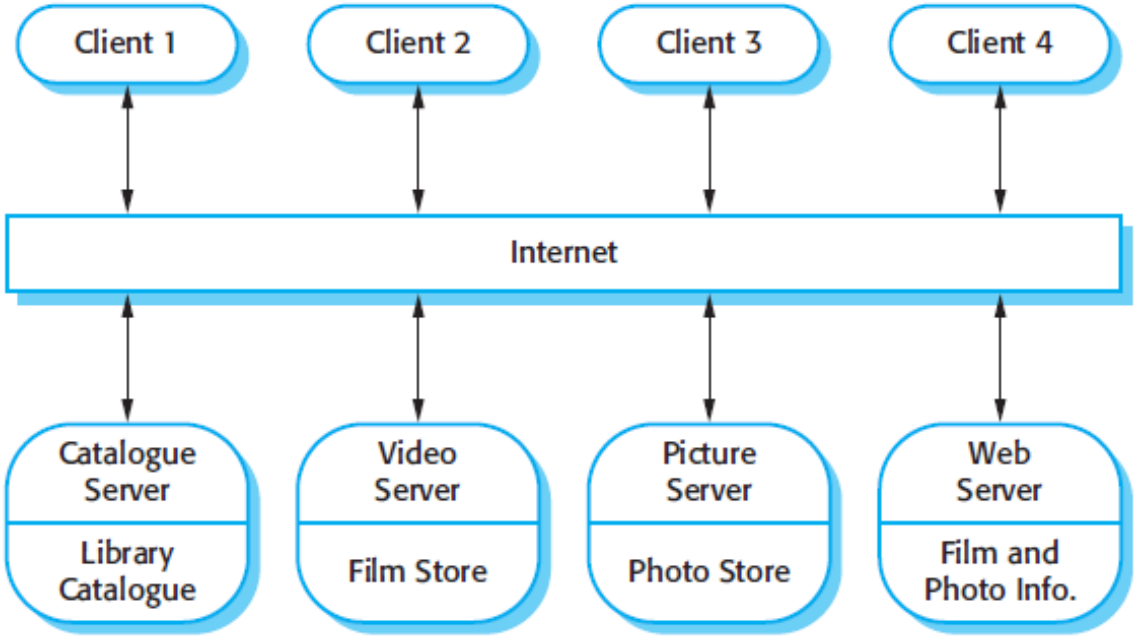
Repository style

- If many components share a lot of data, a repository style might be appropriate
- Components interact by updating the repository
- This pattern is ideal when there is a lot of data stored for a long time
- Good: components can be independent
- Bad: the repository is a single point of failure



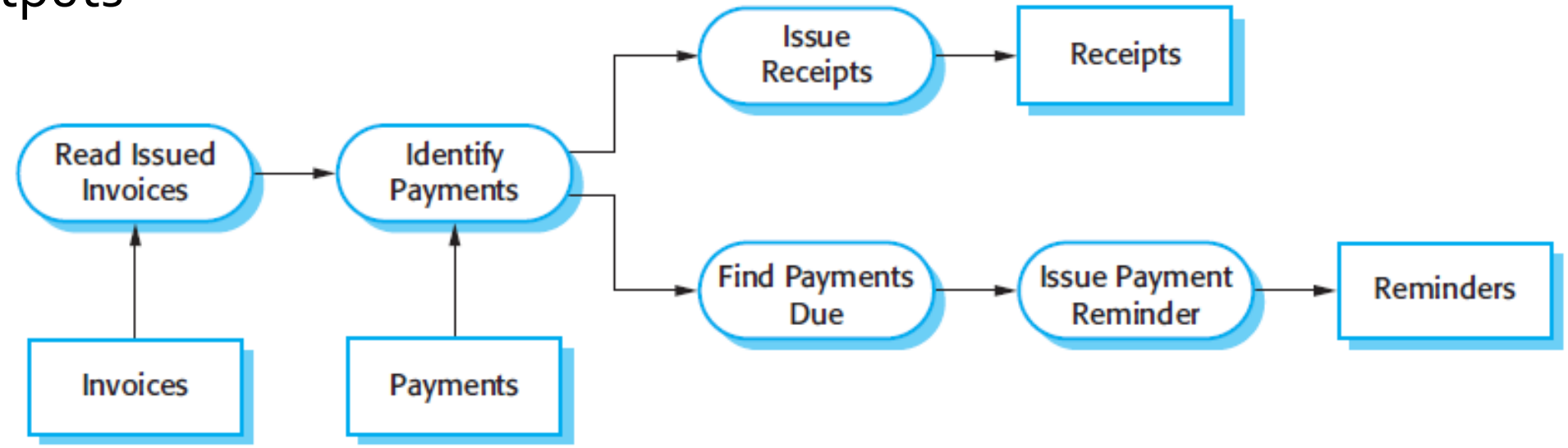
Client-server architecture

- Client-Server styles are used for distributed systems
- Each server provides a separate service, and clients access those services
- Good: work is distributed, and clients can access just what they need
- Bad: each service is a single point of failure, and performance might be unpredictable



Pipe and filter style

- In the pipe and filter style, data is passed from one component to the next
- Each component transforms input into output
- Good: easy to understand, matches business applications, and allows for component reuse
- Bad: each component has to agree on formatting with its inputs and outputs



Performance goals

- Non-functional requirements usually relate to properties of the whole system
- They must be planned for at the architectural level
- Strategies for architecture designers meeting performance requirements:
  - **Use budgets:** Each component is assigned a limit on time, space, or energy so that everything stays within the overall limit
  - **Choose appropriate styles:** Layered architectures with a lot of layers can be slow
  - **Modify styles:** Basic styles can be changed a little, for example, allowing a low-level layer to talk directly to a high-level one
  - **Emphasize simplicity:** Simple versions of components might be faster and use less memory than full-featured ones
  - **Minimize synchronous interactions:** Synchronous interactions are ones where components have to wait for answers from other components

Reliability goals

- **Reliability** is the probability that a product will behave as it should under normal conditions for a given period of time
- Like performance, it must be planned for at the architectural level
- Strategies for architecture designers meeting reliability requirements:
  - **Choose appropriate styles:** For example, client-server is very reliable, as long as the server is reliable
  - **Modify styles:** A client server model could add redundant servers
  - **Control component interactions:** Interactions between components should only happen through explicit interfaces
  - **Handle exceptions:** Plan how exceptions will be handled at the architectural level so that it's clear what parts of the system are responsible for recovering from which errors
  - **Monitor system health:** Build in a component to check to see if other components have failed

Quiz

Upcoming

Next time…

- Detailed design
- Design patterns

Reminders

- Keep reading Chapter 7: Software Engineering Design for Monday
- Keep working on the draft of Project 2
  - Due Friday of next week