

O'REILLY®

Head First

Software Architecture

A Learner's Guide to
Architectural Thinking

Raju Gandhi,
Mark Richards
& Neal Ford



A Brain-Friendly Guide

1

software architecture demystified

Let's Get Started!

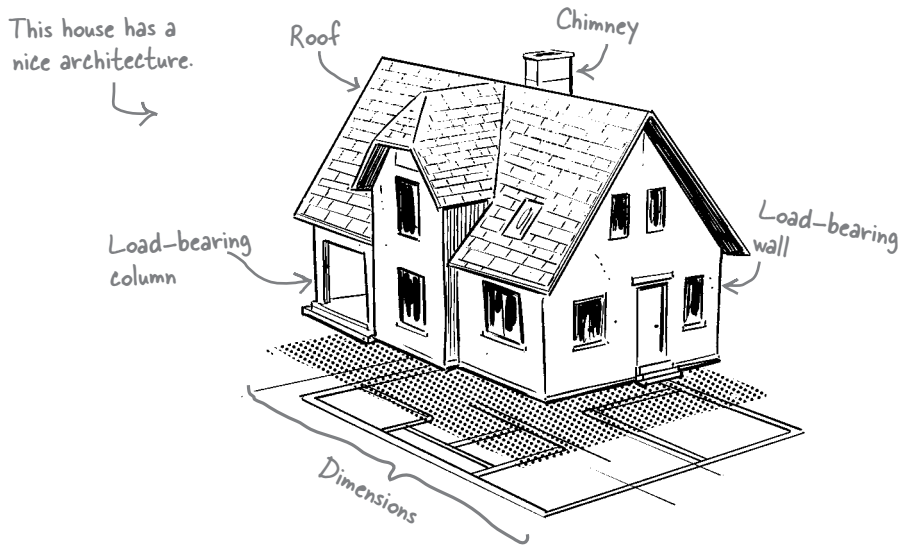


Software architecture is fundamental to the success of your system. This chapter demystifies software architecture. You'll gain an understanding of architectural dimensions and the differences between architecture and design. Why is this important? Because understanding and applying architectural practices helps you build more effective and correct software systems—systems that not only function better, but also meet the needs and concerns of the business and continue to operate as your business and technical environments undergo constant change. So, without further delay, let's get started.

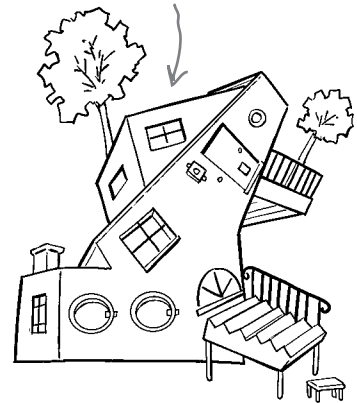
Building your understanding of software architecture

To better understand software architecture, think about a typical home in your neighborhood. The structure of the home is its **architecture**—things like its shape, how many rooms and floors it has, its dimensions, and so on. A house is usually represented through a building plan, which contains all the lines and boxes necessary to know how to build the house. Structural things like those shown below are hard and expensive to change later and are the *important* stuff about the house.

The building metaphor is a very popular one for understanding software architecture.



Not only is this house ugly, it's not very functional either.



Architecture is essential for building a house. Can you imagine building one without an architecture? It might turn out looking something like the house on the right.

Architecture is also essential for building software systems. Have you ever come across a system that doesn't scale, or is unreliable or difficult to maintain? It's likely not enough emphasis was placed on that system's architecture.



Exercise

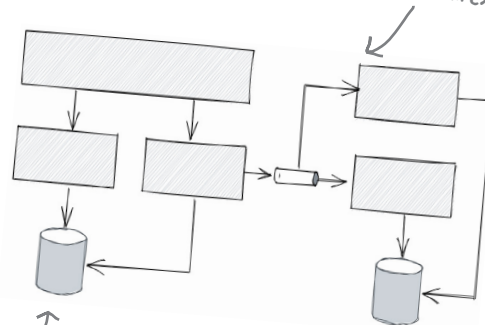
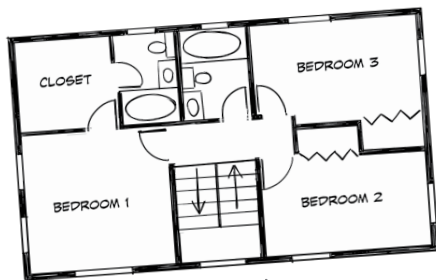
Gardening is another useful metaphor for describing software architecture. Using the space below, can you describe how planning a garden might relate to software architecture? You can see what we came up with at the end of this chapter.

→ Solution on page 29

Building plans and software architecture

You might be wondering how the building plans of your home relate to software architecture. Each is a representation of the thing being built. So what does the “building plan” of a software system look like? Lines and boxes, of course.

A building plan specifies the structure of your home—the rooms, walls, stairs, and so on—in the same way a software architecture diagram specifies its structure (user interfaces, services, databases, and communication protocols). Both artifacts provide guidelines and constraints, as well as a vision of the final result.



Both of these diagrams represent building plans.

Fun fact—a building plan used to be called a “blueprint,” but that term is now obsolete (at least within building architecture).

Sharpen your pencil

What features of your home can you list that are *structural* and related to its *architecture*? You can find our thoughts at the end of this chapter.

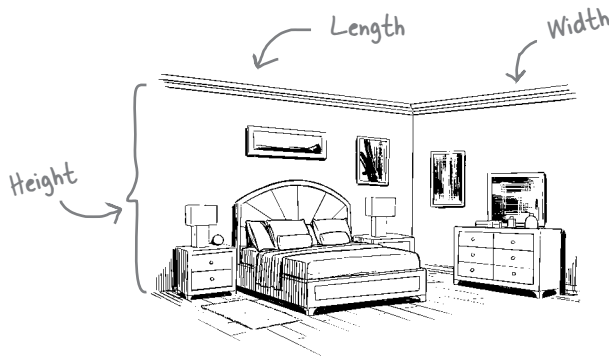
Use this space to write down your ideas.

Did you notice that the floor plan for the house above doesn't specify the details of the rooms—things like the type of flooring (carpet or hardwood), the color of the walls, and where a bed might go in a bedroom? That's because those things aren't *structural*. In other words, they don't specify something about the *architecture* of the house, but rather about its *design*.

Don't worry—you'll learn a lot more about this distinction later in this chapter. Right now, just focus on the structure of something—in other words, its architecture.

→ Solution on page 29

The dimensions of software architecture



Most things around us are multidimensional. For example, you might describe a particular room in your home by saying it is 5 meters long and 4 meters wide, with a ceiling height of 2.5 meters. Notice that to properly describe the room you needed to specify all three dimensions—its height, length, and width.

You can describe software architecture by its dimensions, too. The difference is that software architecture has *four dimensions*.

1 Architectural characteristics

This dimension describes what aspects of the system the architecture needs to support—things like scalability, testability, availability, and so on.

2 Architectural decisions

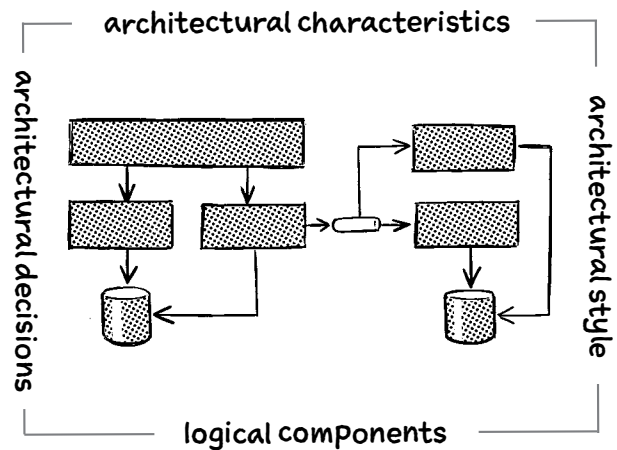
This dimension includes important decisions that have long-term or significant implications for the system—for example, the kind of database it uses, the number of services it has, and how those services communicate with each other.

3 Logical components

This dimension describes the building blocks of the system's functionality and how they interact with each other. For example, an ecommerce system might have components for inventory management, payment processing, and so on.

4 Architectural style

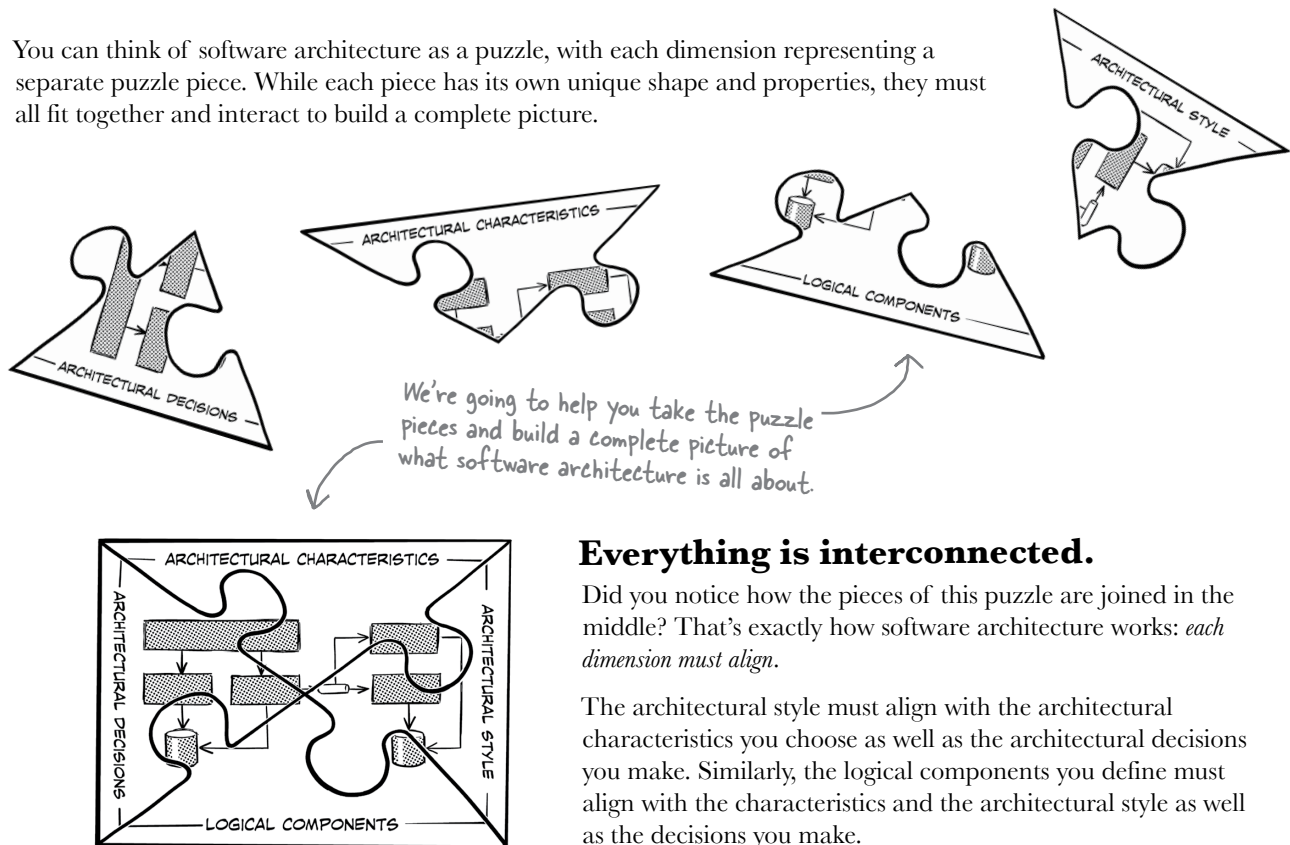
This dimension defines the overall physical shape and structure of a software system in the same way a building plan defines the overall shape and structure of your home.



You'll learn about five of the most common architectural styles later in this book.

Puzzling out the dimensions

You can think of software architecture as a puzzle, with each dimension representing a separate puzzle piece. While each piece has its own unique shape and properties, they must all fit together and interact to build a complete picture.

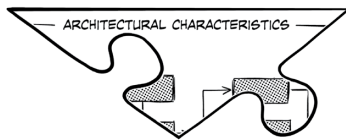


there are no Dumb Questions

Q: Do you need all four dimensions when creating an architecture, or can you skip some if you don't have time?

A: Unfortunately, you can't skip any of these dimensions—they are all required to create and describe an architecture. One common mistake software architects make is using only one or two of these dimensions when describing their architecture. "Our architecture is microservices" describes a single dimension—the architectural style—but leaves too many unanswered questions. For example, what architectural characteristics are critical to the success of the system? What are its logical components (functional building blocks)? What major decisions have you made about how you'll implement the architecture?

The first dimension: Architectural characteristics



Architectural characteristics form the foundation of the architecture in a software system. Without them, you cannot make architectural decisions or analyze important trade-offs.

Imagine you're trying to choose between two homes. One home is roomy but is next to a busy, noisy motorway. The other home is in a nice, quiet neighborhood, but is much smaller. **Which characteristic is more important to you**—home size or the level of noise and traffic? Without knowing that, you can't make the right choice.

The same is true with software architecture. Let's say you need to decide what kind of database to use for your new system. Should it be a relational database, a simple key/value database, or a complex graph database? The answer will be based on what architectural characteristics are critical to you. For example, you might choose a graph database if you need high-speed search capability (we'll call that **performance**), whereas a traditional relational database might be better if you need to preserve data relationships (we'll call that **data integrity**).

performance

The amount of time it takes for the system to process a business request

availability

The amount of uptime of a system; usually measured in "nines" (so 99.9% would be three "nines")

scalability

The system's ability to maintain a consistent response time and error rate as the number of users or requests increases

Here are some of the more common architectural characteristics. You'll be learning all about these in Chapter 2.



Exercise

Check the things you think might be considered architectural characteristics—something that the *structure* of the software system supports.

- Changing the font size in a window on the user interface screen
- Making changes quickly
- Handling thousands of concurrent users
- Encrypting user passwords stored in the database
- Interacting with many external systems to complete a business request

—————> Solution on page 30

The term *architectural characteristics* might not be familiar to you, but that doesn't mean you haven't heard of them before. Collectively, things like performance, scalability, reliability, and availability are also known as nonfunctional requirements, system quality attributes, and simply "the -ilities" because most end with the suffix *-ility*. We like the term *architectural characteristics* because these qualities help define the character of the architecture and what it needs to support.

Architectural characteristics are capabilities that are critical or important to the success of the system.

Make it Stick 

To architect software you must first address:

Capabilities key to the new app's success

Who Does What?

Here's your chance to see how much you already know about many common architectural characteristics. Can you match up each architectural characteristic on the left with its definition on the right? You'll notice there are more definitions than characteristics, so be careful—not all of the definitions have matches.

Extensibility

We did this one for you.

Agility

Interoperability

Fault tolerance

Feasibility

Taking into account time frames, budgets, and developer skills when making architectural choices

The system's ability to keep its other parts functioning when fatal errors occur

The ease with which the system can be enhanced to support additional features and functionality

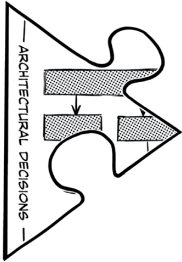
The amount of time it takes to get a response to the user

The system's ability to respond quickly to change (a function of maintainability, testability, and deployability)

The system's ability to interface and interact with other systems to complete a business request

—————▶ **Solution on page 30**

The second dimension: Architectural decisions



Architectural decisions are choices you make about structural aspects of the system that have long-term or significant implications. As constraints, they'll guide your development team in planning and building the system.

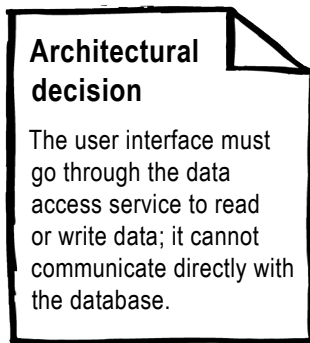
Should your new home have one floor or two? Should the roof be flat or peaked? Should you build a big, sprawling ranch house? These are good examples of architectural decisions because they involve the *structural* aspect of your home.

What should your home look like? This kind of decision is an architectural one.



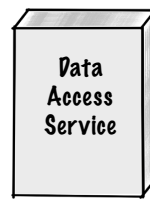
You might decide that your system's user interface should not communicate directly with the database, but instead must go through the underlying services to retrieve and update data. This architectural decision places a particular constraint on the development of the user interface, and also guides the development team about how other components should access and update data in the database.

Here's an example of an architectural decision.



You'll be learning a lot about architectural decisions in Chapter 3.

This architectural decision imposes a constraint and acts as a guide.

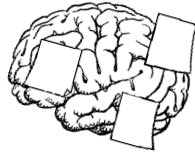


This image represents a service. You'll be seeing it a lot in the book.



This is the database.

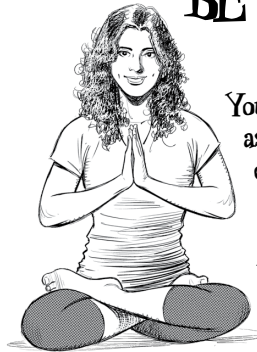
Make it Stick



Decisions are structural guides for dev teams.
They often focus on significant themes.

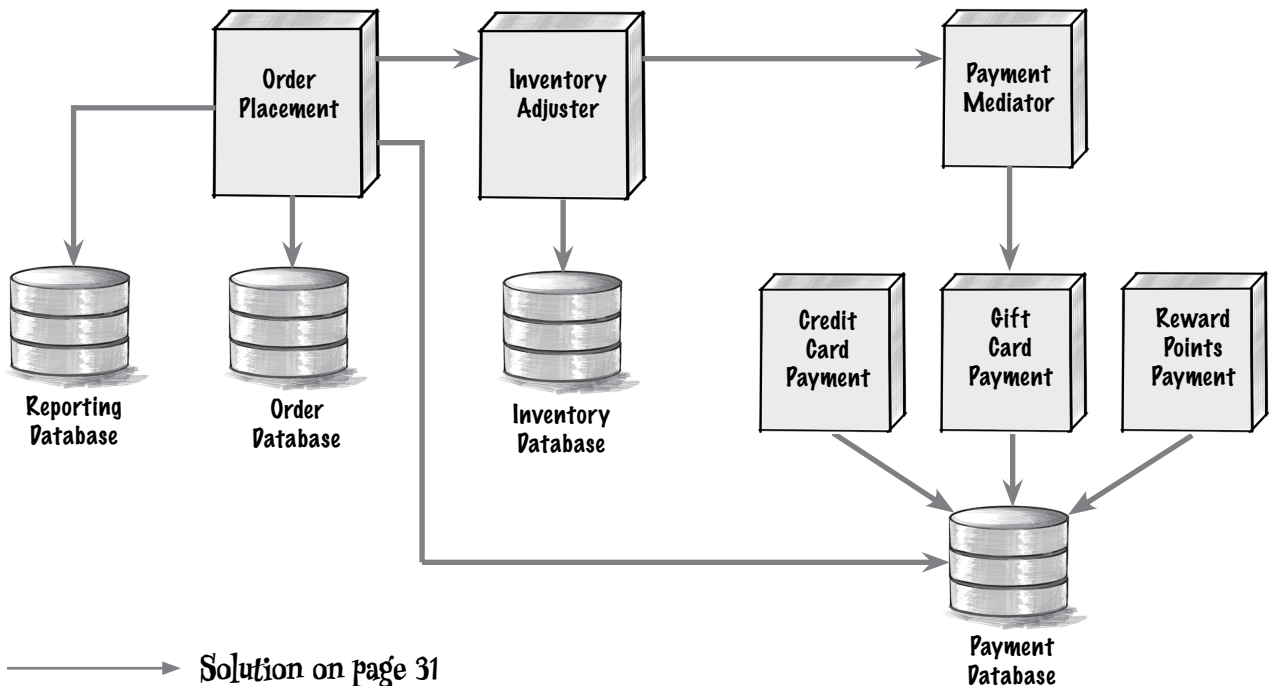
It's not uncommon to have several dozen or more documented architectural decisions within any system. Generally, the larger and more complicated the system, the more architectural decisions it will have.

BE the architect



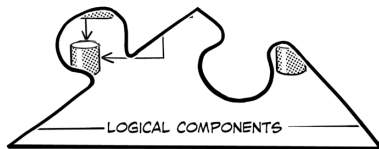
Your job is to be the architect and identify as many architectural decisions as you can in the diagram below. Draw a circle around anything that you think might be an architectural decision and write what that decision might be.

Here's a hint—do you have questions about why certain things are done the way they are?



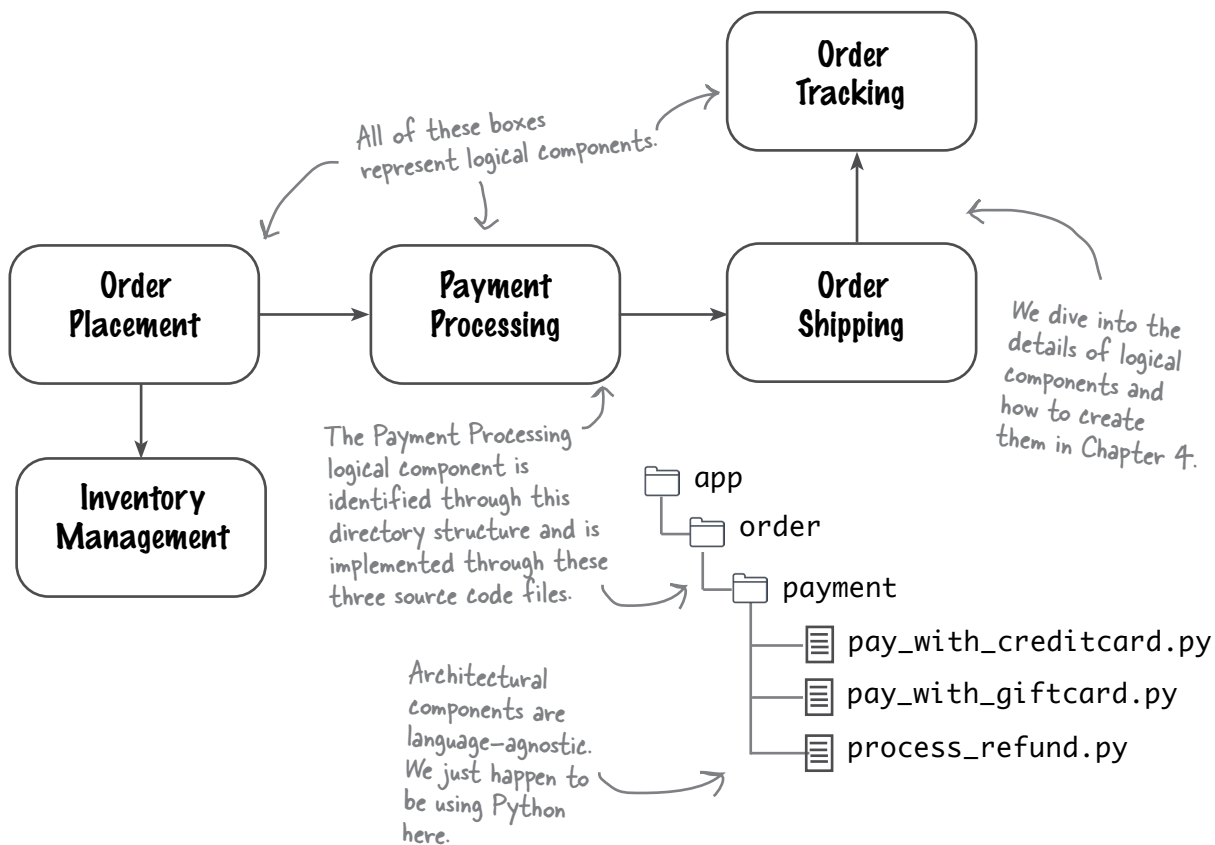
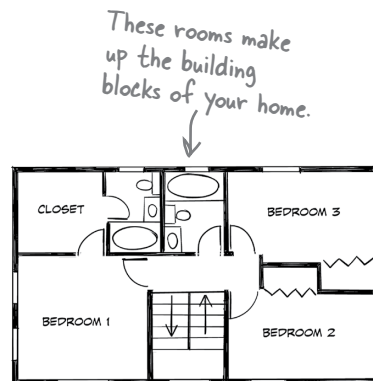
→ Solution on page 31

The third dimension: Logical components



Logical components are the building blocks of a system, much in the same way rooms are the building blocks of your home. A logical component performs some sort of function, such as processing the payment for an order, managing item inventory, or tracking orders.

Logical components in a system are usually represented through a directory or namespace. For example, the directory `app/order/payment` with the corresponding namespace `app.order.payment` identifies a logical component named Payment Processing. The source code that allows users to pay for an order is stored in this directory and uses this namespace.



Sharpen your pencil



You've just created the following two components for a new system, and your development team wants to start writing class files to implement them. Can you create a directory structure for them so they can start coding? Flip to the end of the chapter for our solution.

Customer Profile

Customer Preferences

Use this space to write down your answer.

→ Solution on page 32

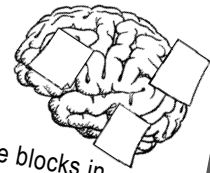
A logical component should always have a well-defined role and responsibility in the system—in other words, a clear definition of what it does.

This component is responsible for “pick and pack.” It locates items in a warehouse (that’s the “pick” part), then determines the correct box size for the items so they can be shipped (that’s the “pack” part).

Order Fulfillment

This is the role and responsibility statement for the Order Fulfillment component.

Make it Stick



Logical components are blocks in conjunction.

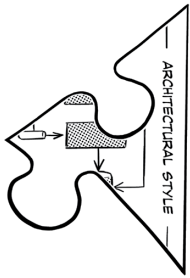
They hold the source code for each business function.

there are no Dumb Questions

Q: What is the difference between the system functionality and the domain?

A: The **domain** is the problem you are trying to solve, and the **system functionality** is how you are solving that problem. In other words, the domain is the “what,” and the system’s functionality is the “how.”

The fourth dimension: Architectural styles



Homes come in all shapes, sizes, and styles. While there are some wild-looking houses out there, most conform to a particular style, such as Victorian, ranch, or Tudor. The style of a home says a lot about its overall structure. For example, ranch homes typically have only one floor; colonial and Tudor homes typically have chimneys; contemporary homes typically have flat roofs.

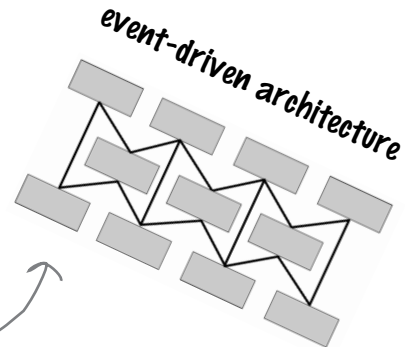
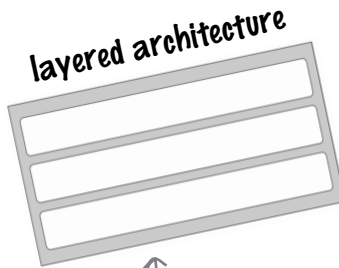
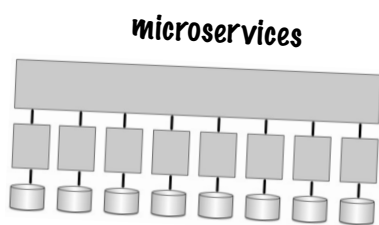
Each region of the world has its own set of home styles—check 'em out at https://en.wikipedia.org/wiki/List_of_house_styles.

What style home do you live in?



Architectural styles define the overall shape and structure of a software system, each with its own unique set of characteristics. For example, the **microservices** architectural style scales very well and provides a high level of *agility*—the ability to respond quickly to change—whereas the **layered** architectural style is less complex and less costly. The **event-driven** architectural style provides high levels of scalability and is very fast and responsive.

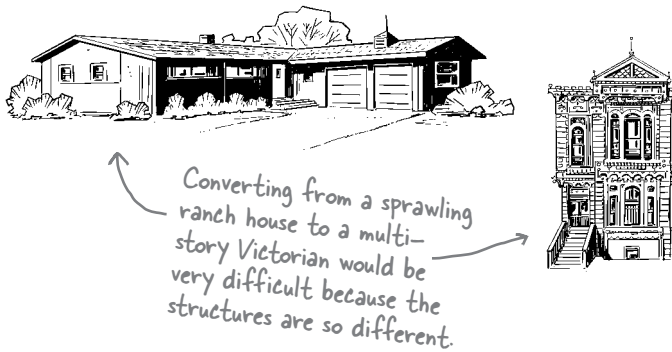
Don't worry—you'll be learning all about these architectural styles later in the book. We've devoted chapters to each of them.



There are a number of different architectural styles, but fortunately not as many as there are house styles.

Because the architectural style defines the overall shape and characteristics of the system, it's important to get it right the first time. Why? Can you imagine starting construction on a one-story ranch home, and in the middle of construction changing your mind and deciding you're going to build a three-story Victorian house instead? That would be a major undertaking, and likely exceed your budget and affect when you can move into the house.

Software architecture is no different. It's not easy changing from a monolithic layered architecture to microservices. Like the house example, this would be quite an undertaking.

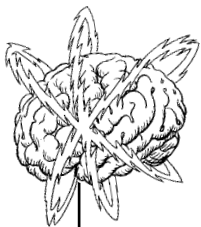


Make it Stick

Styles shape the system and help serve its purposes.
You might choose a monolith or microservices.

Later in the book, we'll show you how to properly select an architectural style based on characteristics that are important to you.

Which brings us back to an earlier point—all of the dimensions of software architecture are interconnected. You can't select an architectural style without knowing what's important to you.



Brain Power



The tightly wound tendons and muscles in a lion's legs enable it to reach speeds as fast as reach speeds as fast as 50 miles (80 kilometers) per hour and leap up to 36 feet (11 meters) in a single bound. This characteristic allows lions to survive by catching fast prey.

Look around you—what else has a structure or shape that defines its characteristics and capabilities?

Fun fact: A lion doesn't have much stamina and can only run fast in short bursts. If you can last longer than the lion chasing you, then you just might survive.

Who Does What?

We were trying to describe our architecture, but all the puzzle pieces got mixed up. Can you help us figure out which dimension does what by matching the statements on the left with the software architecture dimensions on the right? Be careful—some of the statements don't have a match because they are not related to architecture.

This system must be available for our overseas customers.

← This is about availability.

Customers are complaining about the background color of the new user interface.

The product owner insists that we get new features and bug fixes out to our customers as fast as possible.

Our system uses an event-driven architecture.

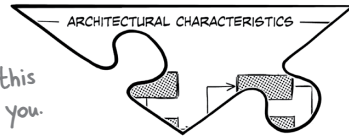
We need to support up to 300,000 concurrent users in this system.

The single payment service will be broken apart into separate services, one for each payment type we accept.

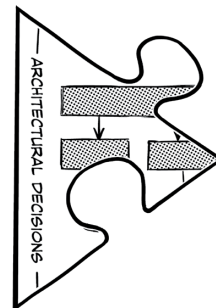
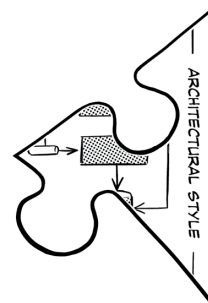
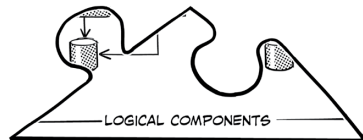
We are going to start offering reward points as a new payment option when paying for an order.

We are breaking up the orderPlacement class into three smaller class files.

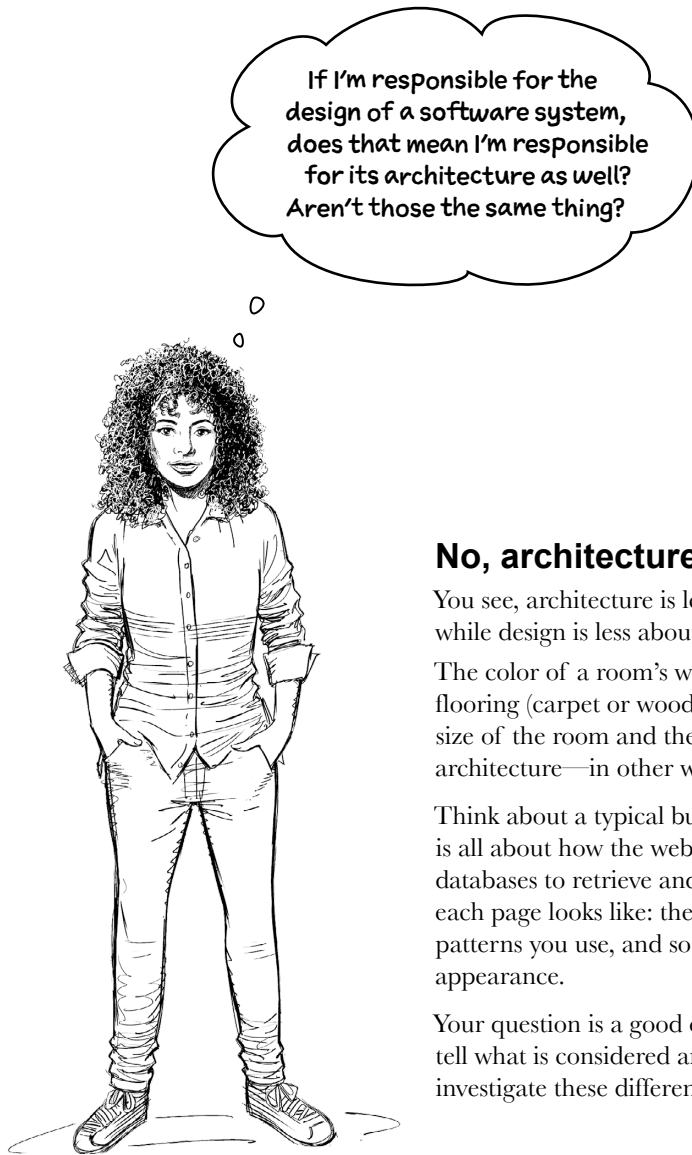
The user interface shall not communicate directly with the database.



↖ We did this one for you.



→ Solution on page 33



If I'm responsible for the design of a software system, does that mean I'm responsible for its architecture as well? Aren't those the same thing?

No, architecture and design are different.

You see, architecture is less about appearance and more about structure, while design is less about structure and more about appearance.

The color of a room's walls, the placement of furniture, and the type of flooring (carpet or wood) are all aspects of design, whereas the physical size of the room and the placement of doors and windows are part of architecture—in other words, the *structure* of the room.

Think about a typical business application. The architecture, or structure, is all about how the web pages communicate with backend services and databases to retrieve and save data, whereas the design is all about what each page looks like: the colors, the placement of the fields, which design patterns you use, and so on. Again, it becomes a matter of structure versus appearance.

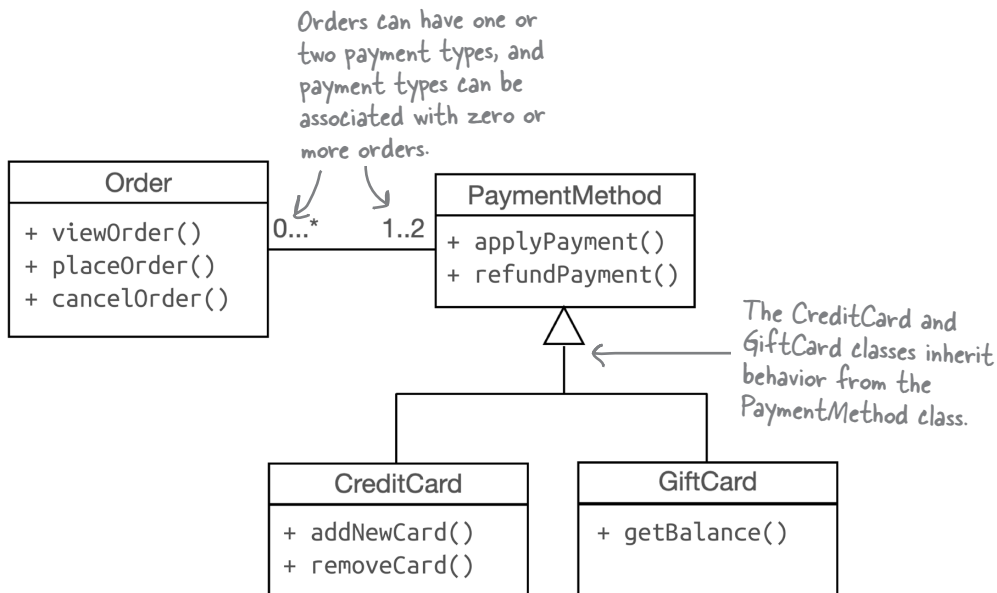
Your question is a good one, because sometimes it gets confusing trying to tell what is considered architecture and what is considered design. Let's investigate these differences.

A design perspective

Suppose your company wants to replace its outdated order processing system with a new custom-built one that better suits its specific needs. Customers can place orders and can view or cancel orders once they have been placed. They can pay for an order using a credit card, a gift card, or both payment methods.



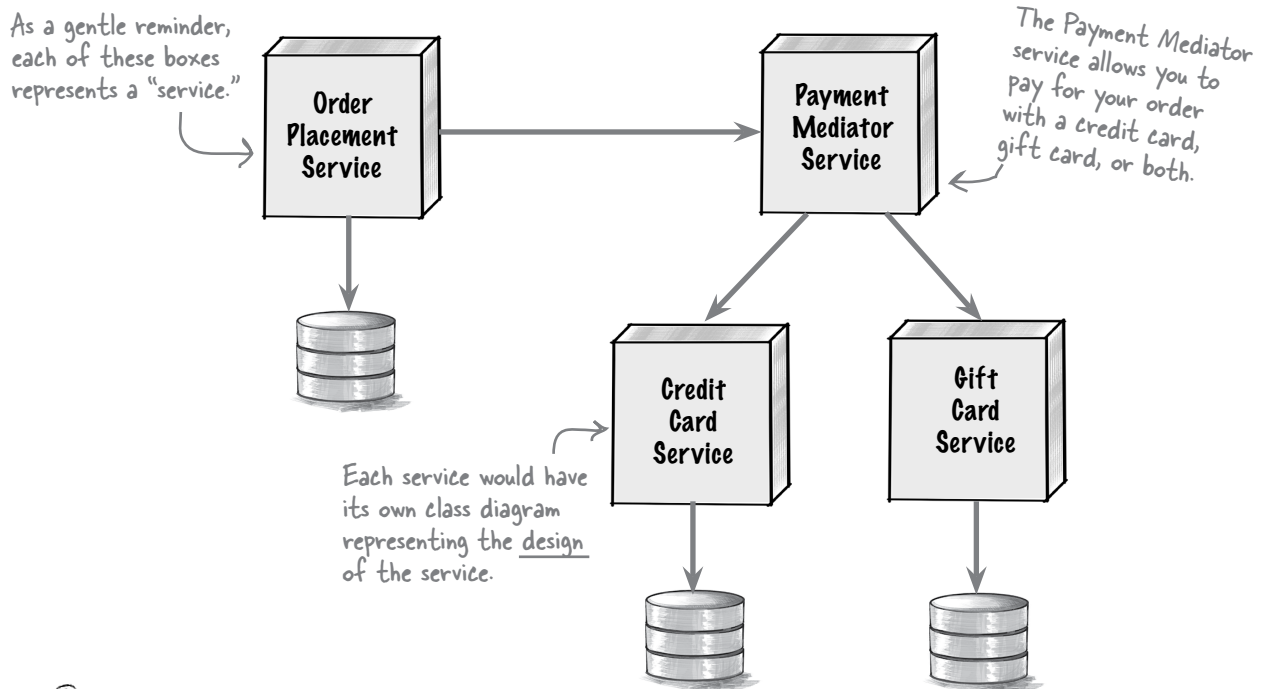
From a *design* perspective, you might build a Unified Modeling Language (UML) class diagram like the one below to show how the classes interact with each other to implement the payment functionality. While you could write source code to implement these class files, this design says nothing about the *physical structure* of the source code—in other words, how these class files would be organized and deployed.



An architectural perspective

Unlike design, architecture is about the *structure* of the system—things like services, databases, and how services communicate with each other and the user interface.

Let's think about that new order processing system again. What would the *system* look like? From an *architectural* perspective, you might decide to create separate services for each payment type within the order payment process and have an orchestrator service to manage the payment processing part of the system, like in the diagram below.



Exercise

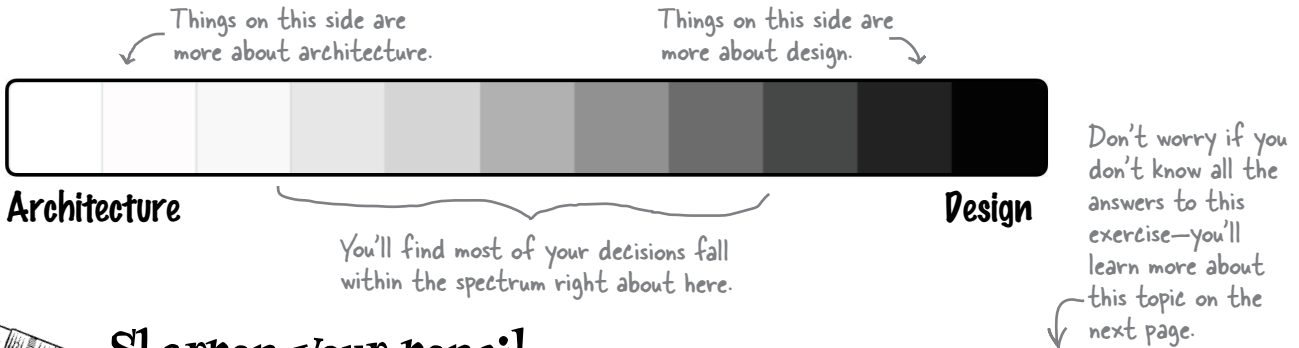
Check all of the things that should be included in a diagram from an *architectural perspective*.

- How services communicate with each other
- The platform and language in which the services are implemented
- Which services can access which databases
- How many services and databases there are

—————> Solution on page 34

The spectrum between architecture and design

Some decisions are certainly architectural (such as deciding which architectural style to use), and others are clearly design-related (such as changing the position of a field on a screen or changing the type of a field within a class). In reality, most decisions you encounter will fall between these two examples, within a *spectrum* of architecture and design.



Sharpen your pencil

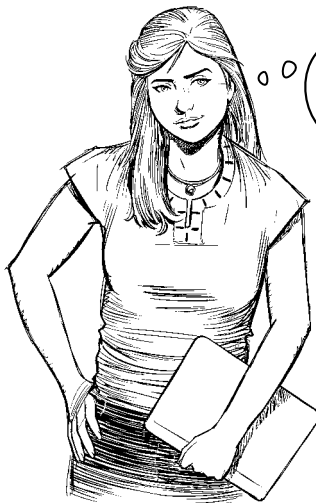
Circle all of the things that you think fall somewhere in the middle of the spectrum *between* architecture and design.

Breaking up a class file Deciding to use a graph database Selecting a user interface framework

Choosing a persistence framework Breaking apart a service Redesigning a web page Migrating to microservices

Choosing an XML parsing library

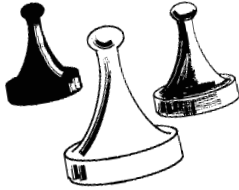
—————> Solution on page 34



Why should I care where in the spectrum between architecture and design my decision lies? Does it really matter that much?

Yes, it matters a lot. You see, knowing where along the spectrum between architecture and design your decision lies helps determine *who* should be responsible for ultimately making that decision. There are some decisions that the development team should make (such as designing the classes to implement a certain feature), some decisions that an architect should make (such as choosing the most appropriate architectural style for a system), and others that should be made together (such as breaking apart services or putting them back together).

Where along the spectrum does your decision fall?



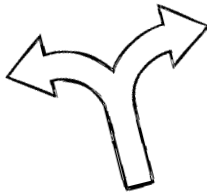
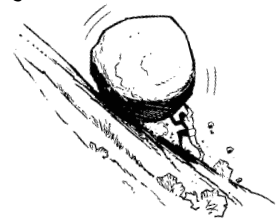
Is it strategic or tactical?

Strategic decisions are long term and influence future actions or decisions. **Tactical** decisions are short term and generally stand independent of other actions or decisions (but may be made in the context of a particular strategy). For example, deciding how big your new home will be influences the number of rooms and the sizes of those rooms, whereas deciding on a particular lighting fixture won't affect decisions about the size of your dining room table. The more strategic the decision, the more it sits toward the architecture side of the spectrum.

Sometimes waking up in the morning requires a lot of effort—we'll call those "architecture" mornings.

How much effort will it take to construct or change?

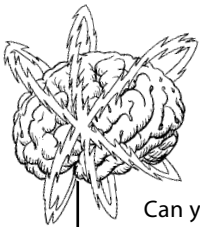
Architectural decisions require more effort to construct or change, while design decisions require relatively less. For example, building an addition to your home generally requires a high level of effort and would therefore be more on the architecture side of the spectrum, whereas adding an area rug to a room requires much less effort and would therefore be more on the design side.



Does it have significant trade-offs?

Trade-offs are the pros and cons you evaluate as you are making a decision. Decisions that involve significant trade-offs require much more time and analysis to make and tend to be more architectural in nature. Decisions that have less-significant trade-offs can be made quicker, with less analysis, and therefore tend to be more on the design side.

We're going to walk you through the details of all three of these factors in the next several pages.

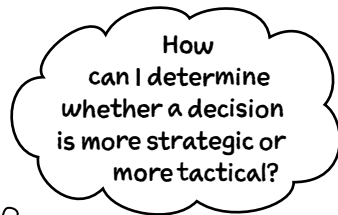


Brain Power

Can you think of a decision that doesn't involve a trade-off, no matter how small or insignificant? Here's a hint: if you think you've found a decision that doesn't involve a trade-off, keep looking.

Strategic versus tactical

The more strategic a decision is, the more architectural it becomes. This is an important distinction, because decisions that are strategic require more thought and planning and are generally long term.



Good question. You can use these three questions to help determine if something is more strategic or tactical. Remember, the more strategic something is, the more it's about architecture.

1. How much thought and planning do you need to put into the decision?

If making the decision takes a couple of minutes to an hour, it's more tactical in nature. If thought and planning require several days or weeks, it's likely more strategic (hence more architectural).

2. How many people are involved in the decision?

The more people involved, the more strategic the decision. A decision you can make by yourself or with a colleague is likely to be tactical. A decision that requires many meetings with lots of stakeholders is probably more strategic.

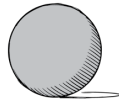
3. Does your decision involve a long-term vision or a short-term action?

If you are making a quick decision about something that is temporary or likely to change soon, it's more tactical and hence more about design. Conversely, if this is a decision you'll be living with for a very long time, it's more strategic and more about architecture.

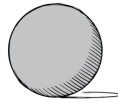
Sharpen your pencil



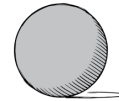
Oh dear. We've lost all of our marbles and we need your help collecting them and putting them back in the right spot. Using the three questions on the previous page as a guide, can you figure out which jar each marble should go in?



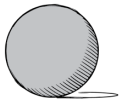
Picking a programming language for your new project



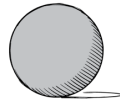
Deciding to get your first dog



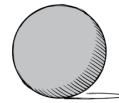
Deploying in the cloud or on premises



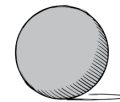
Redesigning your user interface



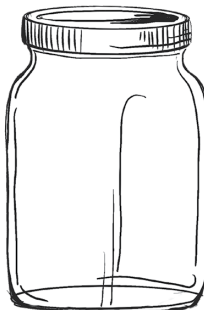
Migrating your system to microservices



Choosing a parsing library



Using a design pattern



Strategic



Somewhere in between



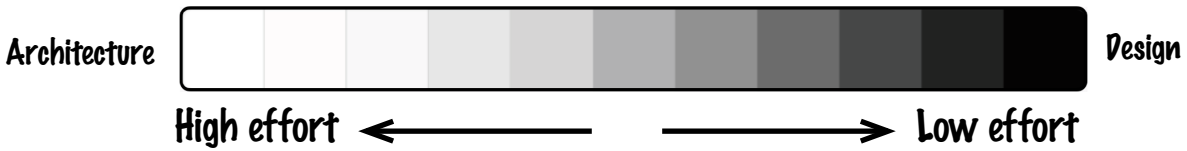
Tactical

→ **Solution on page 35**

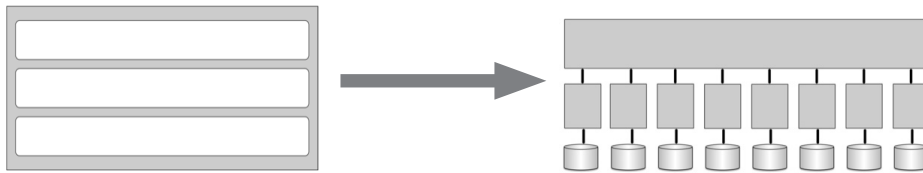
High versus low levels of effort

Renowned software architect and author Martin Fowler once wrote that “software architecture is the stuff that’s hard to change.” You can use Martin’s definition to help determine where along the spectrum your decision lies. The harder something is to change later, the further it falls toward the architecture side of the spectrum. Conversely, the easier it is to change later, the more it’s probably related to design.

Martin Fowler’s website (<https://martinfowler.com/architecture>) has lots of useful stuff about architecture.



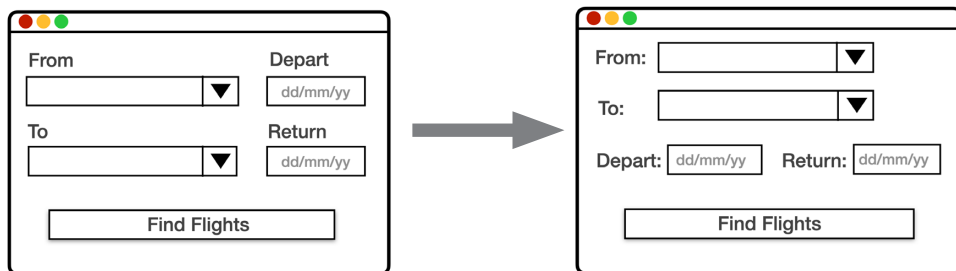
Suppose you are planning on moving from one architectural style to another; say, from a traditional *n*-tiered layered architecture to microservices. This migration effort is rather difficult and will take a lot of time. Because the level of effort is high, this would be on the far end of the *architecture side* of the spectrum.

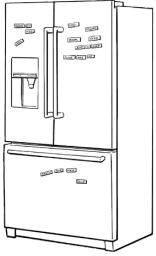


Oh dear, this is going to take a lot of effort. Changing things that are architectural is difficult.

Changing the layout of the fields on a web page is more about appearance than structure—yet another reason why this would be considered design.

Now suppose you’re rearranging fields on a user interface screen. This task takes relatively less effort, so it resides on the far end of the *design side* of the spectrum.





Code Magnets

We had all of these magnets from our to-do list arranged from high effort to low effort, and somehow they all fell on the floor and got mixed up. Can you help us put them back in the right order based on the amount of effort it would take to make each change?

Draw arrows to put the high-effort tasks at the top of the page, and the lower-effort ones toward the bottom of the page.

Resolving a merge conflict in Git

Replacing your user interface framework

Migrating your system to a cloud environment

Deciding which mustard to buy

Renaming a method or function

Breaking apart a single service into separate ones

Moving from a relational to a graph database

Breaking apart a class file

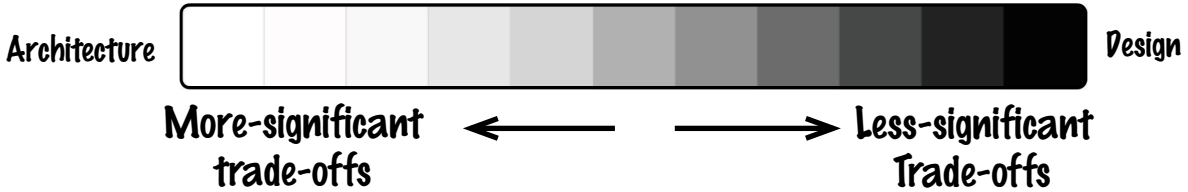
High effort

Low effort

→ Solution on page 36

Significant versus less-significant trade-offs

Some decisions you make might involve significant trade-offs, such as choosing which city to live in. Others might involve less significant trade-offs, like deciding on the color of your living room rug. You can use the level of significance of the trade-offs in a particular decision to help determine whether that decision is more about architecture or design. The more significant the trade-offs, the more it's about architecture; the less significant the trade-offs, the more it's about design.



I wonder if microservices might be a good fit for this project.

- + Scalability
- + Agility
- + Elasticity
- + Fault tolerance



- Cost
- Complexity
- Performance
- Workflow

Wow—these are some serious trade-offs to consider. This is more about architecture.



Should I break my class file apart?

- + Maintainability
- + Readability



- More classes

This trade-off is not so significant, making this decision more about design.



Exercise

Decisions, decisions, decisions. How can we ever tackle all of these decisions? One thing we think might help is to identify the decisions that involve significant trade-offs, since those will require more thinking and will take longer. Can you help us by identifying which decisions have significant trade-offs and which don't?

Is this a significant trade-off?

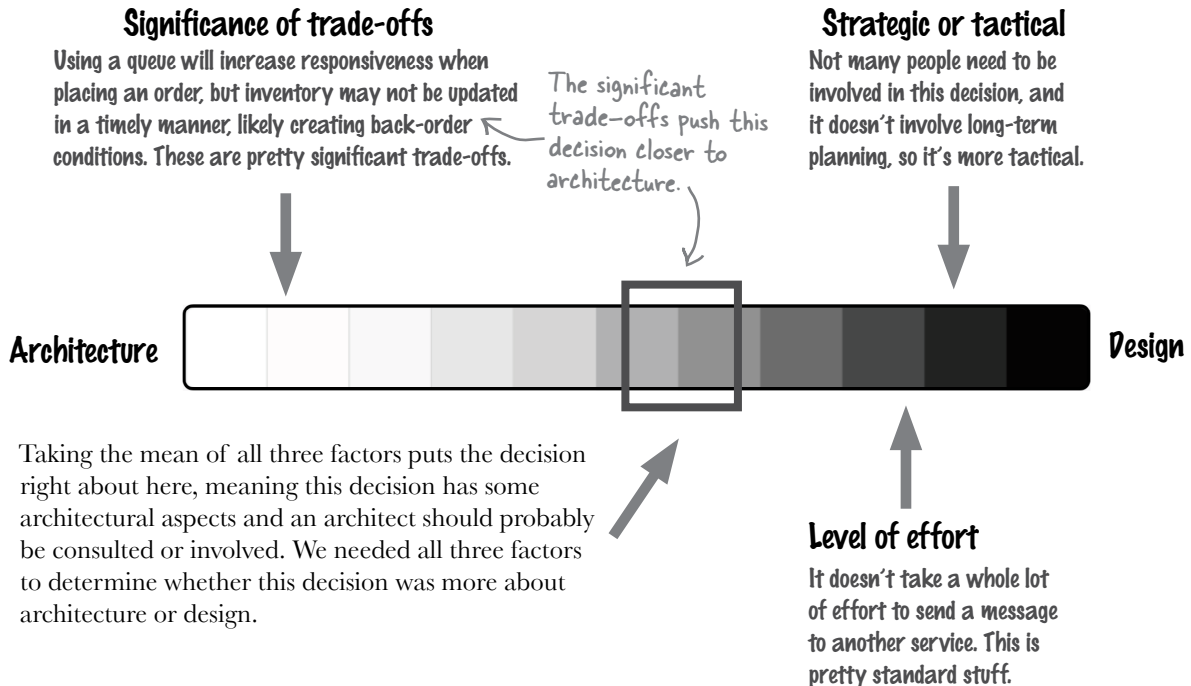
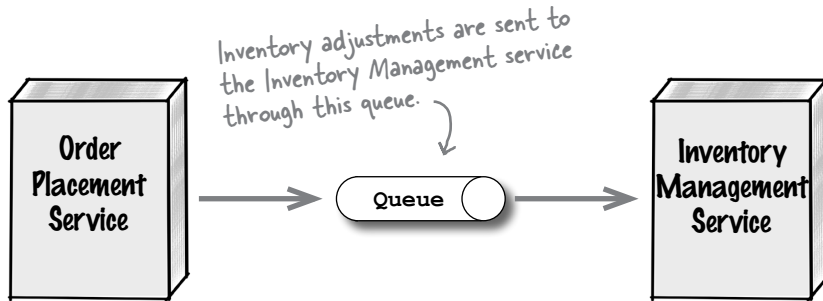
- | | | |
|------------------------------|-----------------------------|---|
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Picking out what clothes to wear to work today |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Choosing to deploy in the cloud or on premises |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Selecting a user interface framework |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Naming a variable in a class file |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Choosing between vanilla and chocolate ice cream |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Deciding which architectural style to use |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Choosing between REST and messaging |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Using full data or only keys for the message payload |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Selecting an XML parsing library |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Deciding whether or not to break apart a service |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Choosing between atomic or distributed transactions |
| <input type="checkbox"/> Yes | <input type="checkbox"/> No | Deciding whether or not to go out to dinner tonight |

—————▶ **Solution on page 37**

Putting it all together

Now it's time to put all three of these factors to use to figure out whether a decision is more about architecture or more about design. This tells development teams when to collaborate with an architect and when to make a decision on their own.

Let's say you decide to use asynchronous messaging between the Order Placement service and the Inventory Management service to increase the system's responsiveness when customers place orders. After all, why should the customer have to wait for the business to adjust and process inventory? Let's see if we can determine where in the spectrum this decision lies.



You made it!

Congratulations—you made it through the first part of your journey to understanding software architecture. But before you roll up your sleeves to dig into further chapters, here’s a little quiz for you to test your knowledge so far. For each of the statements below, circle whether it is true or false.

True or False

- | | | |
|------|-------|--|
| True | False | Design is like the structure of a house (walls, roof, layout, and so on), and software architecture is like the furniture and decoration. |
| True | False | Most decisions are purely about architecture or design. Very few exist along a spectrum between architecture and design. |
| True | False | The more strategic your decision, the more it’s about architecture; the more tactical, the more it’s about design. |
| True | False | The more effort it takes to implement or change your decision, the more it’s about design; the less effort, the more it’s about architecture. |
| True | False | <i>Trade-offs</i> are the pros and cons of a given decision or task. The more significant the trade-offs become, the more it’s about architecture. |

—————▶ **Solution on page 38**

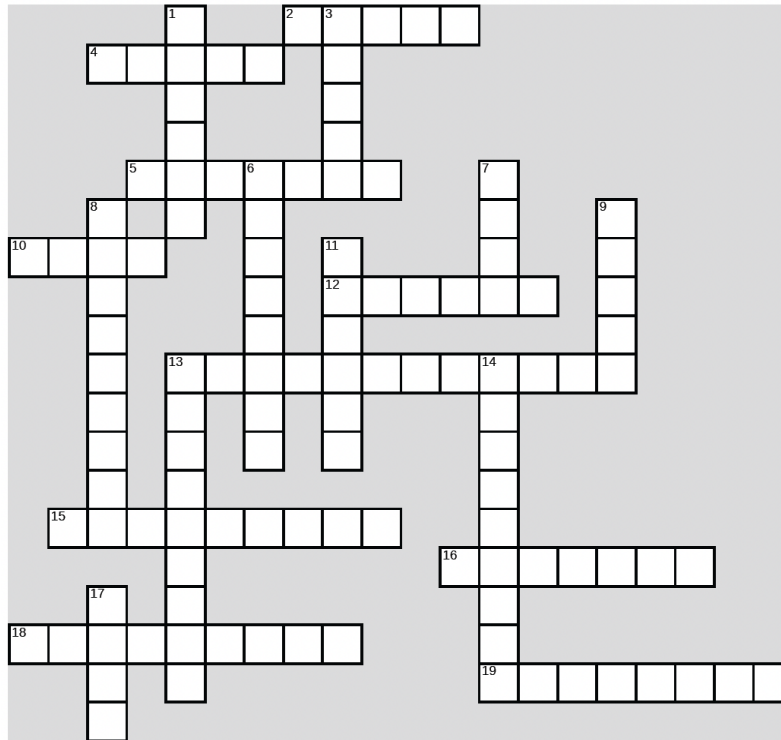
Bullet Points

- Software architecture is less about appearance and more about structure, whereas design is more about appearance and less about structure.
- You need to use four dimensions to understand and describe software architecture: architectural characteristics, architectural decisions, logical components, and architectural style.
- Architectural characteristics form the foundational aspects of software architecture. You must know which architectural characteristics are most important to your specific system, so you can analyze trade-offs and make the right architectural decisions.
- Architectural decisions serve as guideposts to help development teams understand the constraints and conditions of the architecture.
- The logical components of a software architecture solution make up the building blocks of the system. They represent things the system does and are implemented through class files or source code.
- Like with houses, with software there are many different architectural styles you can use. Each style supports a specific set of architectural characteristics, so it’s important to make sure you select the right one (or combination of them) for your system.
- It’s important to know if a decision is about architecture or design, because that helps determine who should be responsible for the decision and how important it is.



Software Architecture Crossword

Congratulations! You made it through the first chapter and learned about what software architecture is (and isn't). Now, why don't you try architecting the solution to this crossword?



Across

2. An architectural style determines the system's overall _____
4. _____-driven is an architectural style
5. Architectural characteristics are sometimes called this
10. Architectural decisions are usually _____ term
12. If something takes a lot of _____ to implement, it's probably architectural
13. You're learning about software _____
15. You'll make lots of architectural _____
16. A system's _____ components are its building blocks
18. The number of rooms in your home is part of its _____
19. Architecture and design exist on a _____

Down

1. Strategic decisions typically involve a lot of these
3. Building this can be a great metaphor
6. Decisions can be strategic of _____
7. How many dimensions it takes to describe a software architecture
8. A website's user _____ involves lots of design decisions
9. The overall shape of a house or a system, like Victorian or microservices
11. It's important to know whether a decision is about architecture or this
13. You might want to become one after reading this book
14. You analyze these when making an architectural decision
17. Trade-offs are about the _____ and cons

—————> **Solution on page 39**



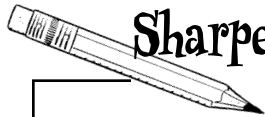
Exercise Solution

From page 2

Gardening is another useful metaphor for describing software architecture. Using the space below, can you describe how a garden might relate to software architecture?

The overall layout of a garden can be compared to the architectural style, whereas each grouping of like plants (either by type or color) can represent the architectural components. Individual plants within a group represent the class files implementing those components.

Gardens are influenced by weather in the same way a software architecture is influenced by changes in technology, platforms, the deployment environment, and so on. Also, if you don't pay attention to the garden, weeds grow—just like structural decay within your architecture.



Sharpen your pencil Solution

From page 3

What features of your home can you list that are *structural* and related to its *architecture*?

The size and shape of your kitchen (who doesn't complain about how small their kitchen is?)

How many floors it has (as you get older, stairs might be a problem)

Where the front door is and if the entranceway is wheelchair accessible

The size of your bedroom closet (if you have lots of clothes)

The height of your ceilings (especially if you happen to be very tall)

How many bathrooms it has (adding a new bathroom is really hard to do)

An attic for storing all of the stuff you never use

Outside deck or patio (unless you live in the Arctic, of course)



Exercise Solution

From page 6

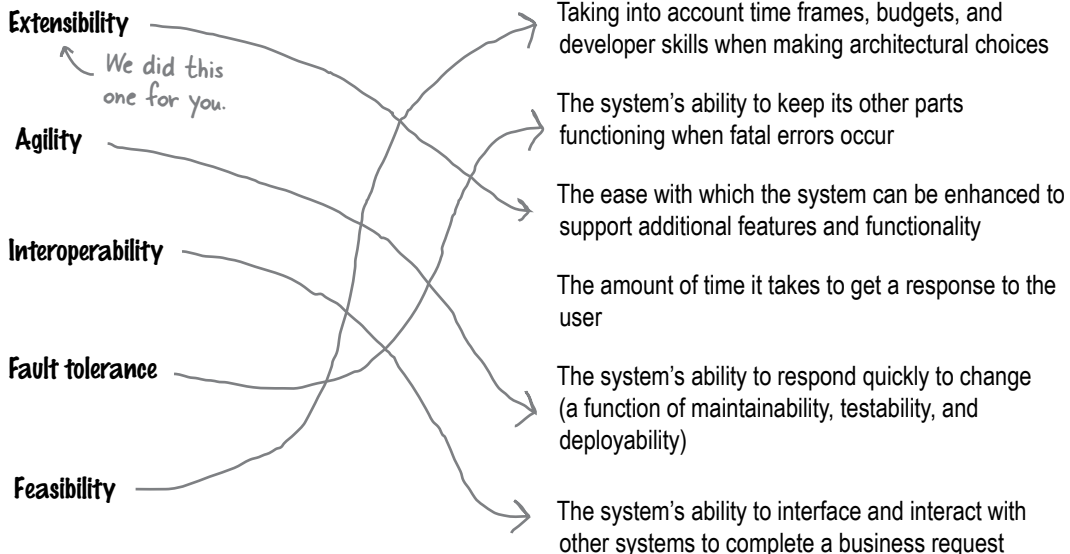
Check the things you think might be considered architectural characteristics—something that the *structure* of the software system supports.

- Changing the font size in a window on the user interface screen
- Making changes quickly ← This is known as agility in architecture.
- Handling thousands of concurrent users ← This is known as elasticity.
- Encrypting user passwords stored in the database
- Interacting with many external systems to complete a business request ← This is known as interoperability.

Who Does What? Solution

From page 7

Here's your chance to see how much you already know about many common architectural characteristics. Can you match up each architectural characteristic on the left with its definition on the right? You'll notice there are more definitions than characteristics, so be careful—not all of the definitions have matches.

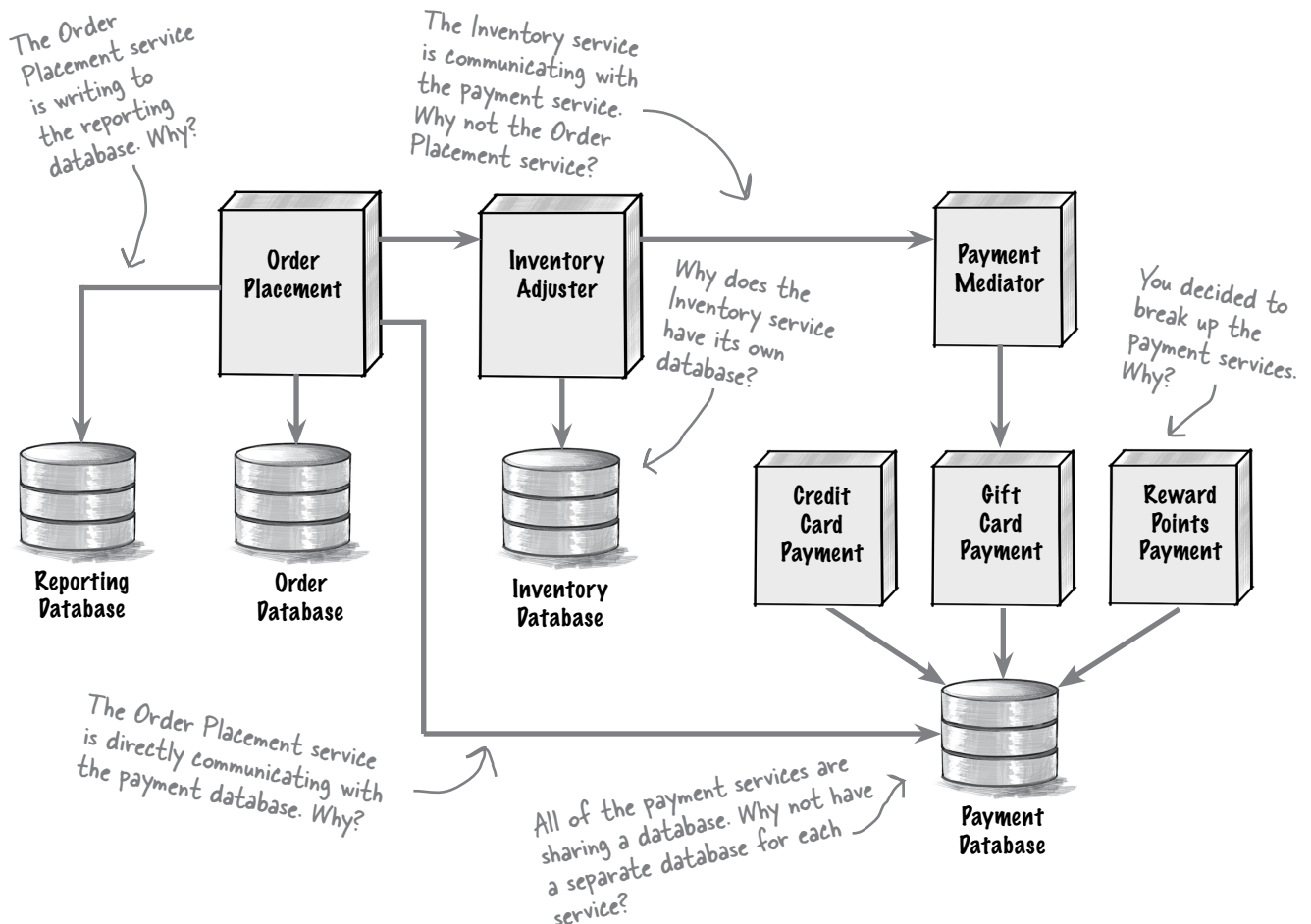


From page 9

BE the architect Solution



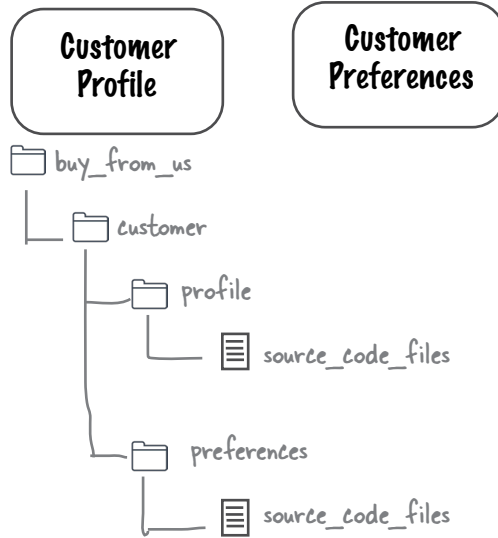
Your job is to be the architect and identify as many architectural decisions as you can in the diagram below. Draw a circle around anything that you think might be an architectural decision and write what that decision might be.





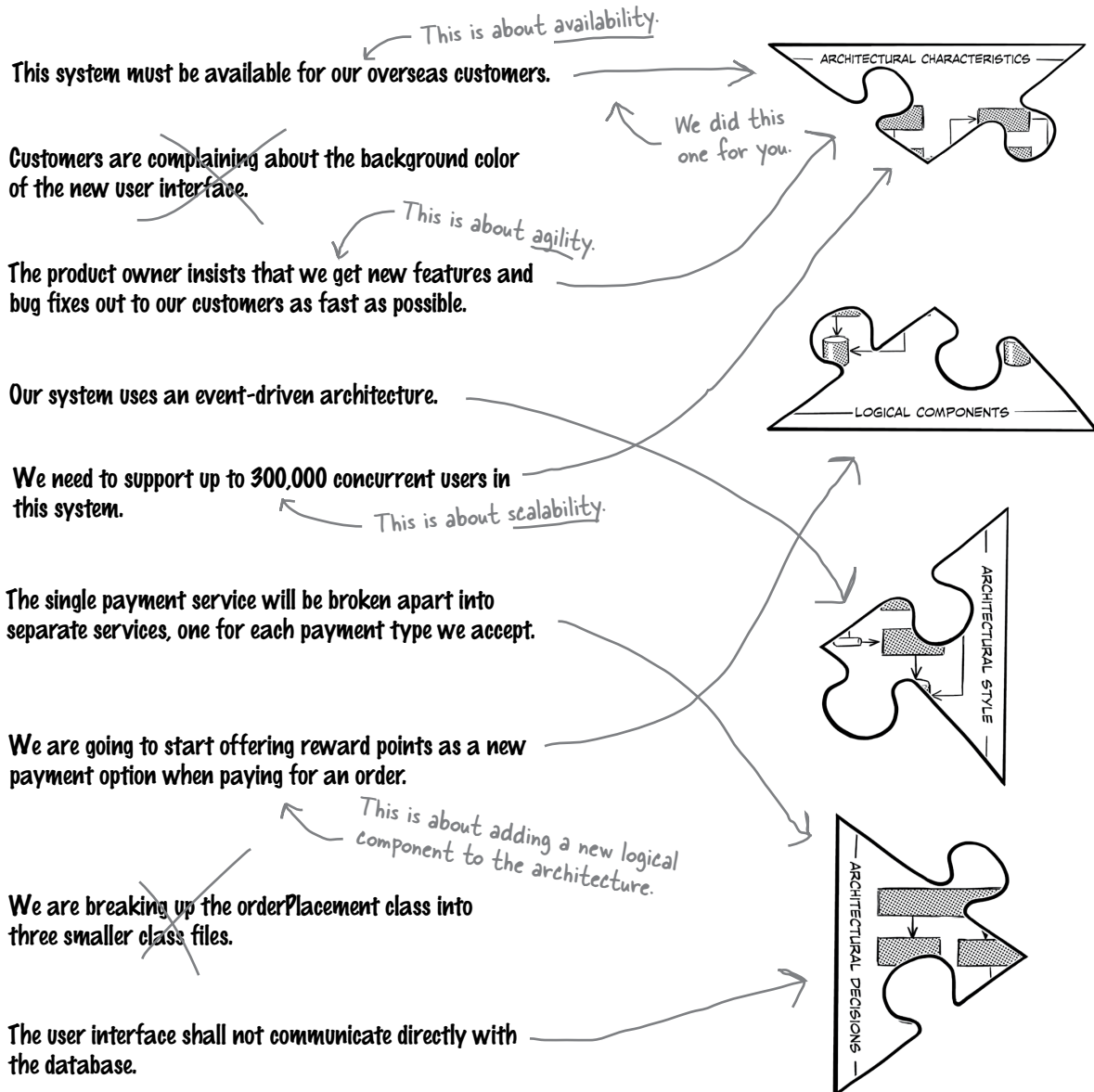
Sharpen your pencil Solution

You've just created the following two components for a new system, and your development team wants to start writing class files to implement them. Can you create a directory structure for them so they can start coding?



Who Does What? Solution

We were trying to describe our architecture, but all the puzzle pieces got mixed up. Can you help us figure out which dimension does what by matching the statements on the left with the software architecture dimensions on the right? Be careful—some of the statements don't have a match because they are not related to architecture.





Exercise Solution

From page 17

Check all of the things that should be included in a diagram from an *architectural perspective*.

- How services communicate with each other
- The platform and language in which the services are implemented
- Which services can access which databases
- How many services and databases there are

How something should be implemented is a design perspective.



Sharpen your pencil Solution

From page 18

Circle all of the things that you think fall somewhere in the middle of the spectrum *between* architecture and design.

This is design.

Breaking up a class file

Deciding to use a graph database

Choosing a persistence framework

Breaking apart a service

Redesigning a web page

Selecting a user interface framework

Migrating to microservices

Choosing an XML parsing library

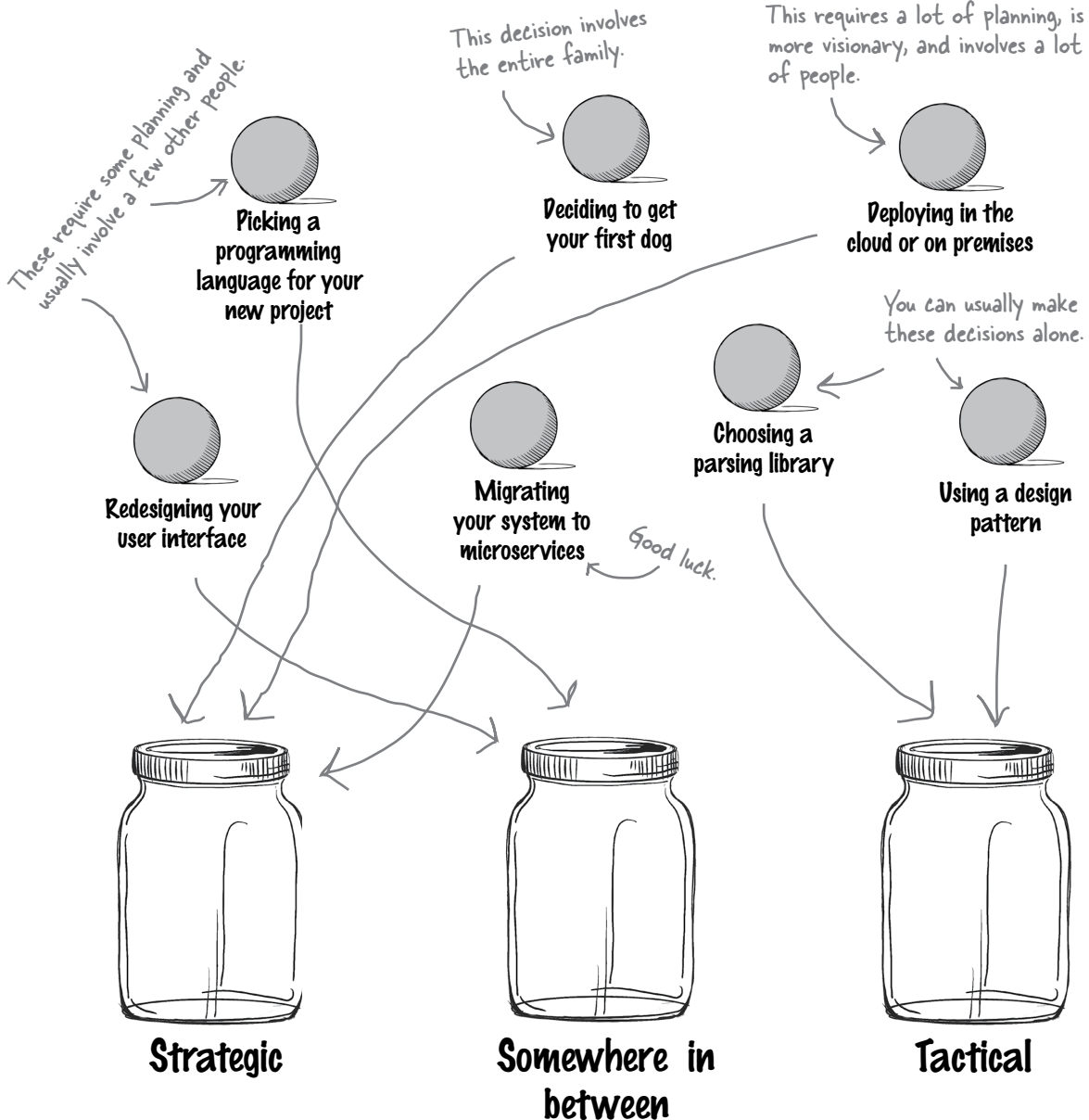
These are design.

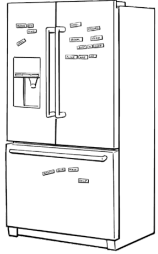
This is architecture.

From page 21

Sharpen your pencil Solution

Oh dear. We've lost all of our marbles and we need your help collecting them and putting them back in the right spot. Using the three questions on page 20 as a guide, can you figure out which jar each marble should go in?

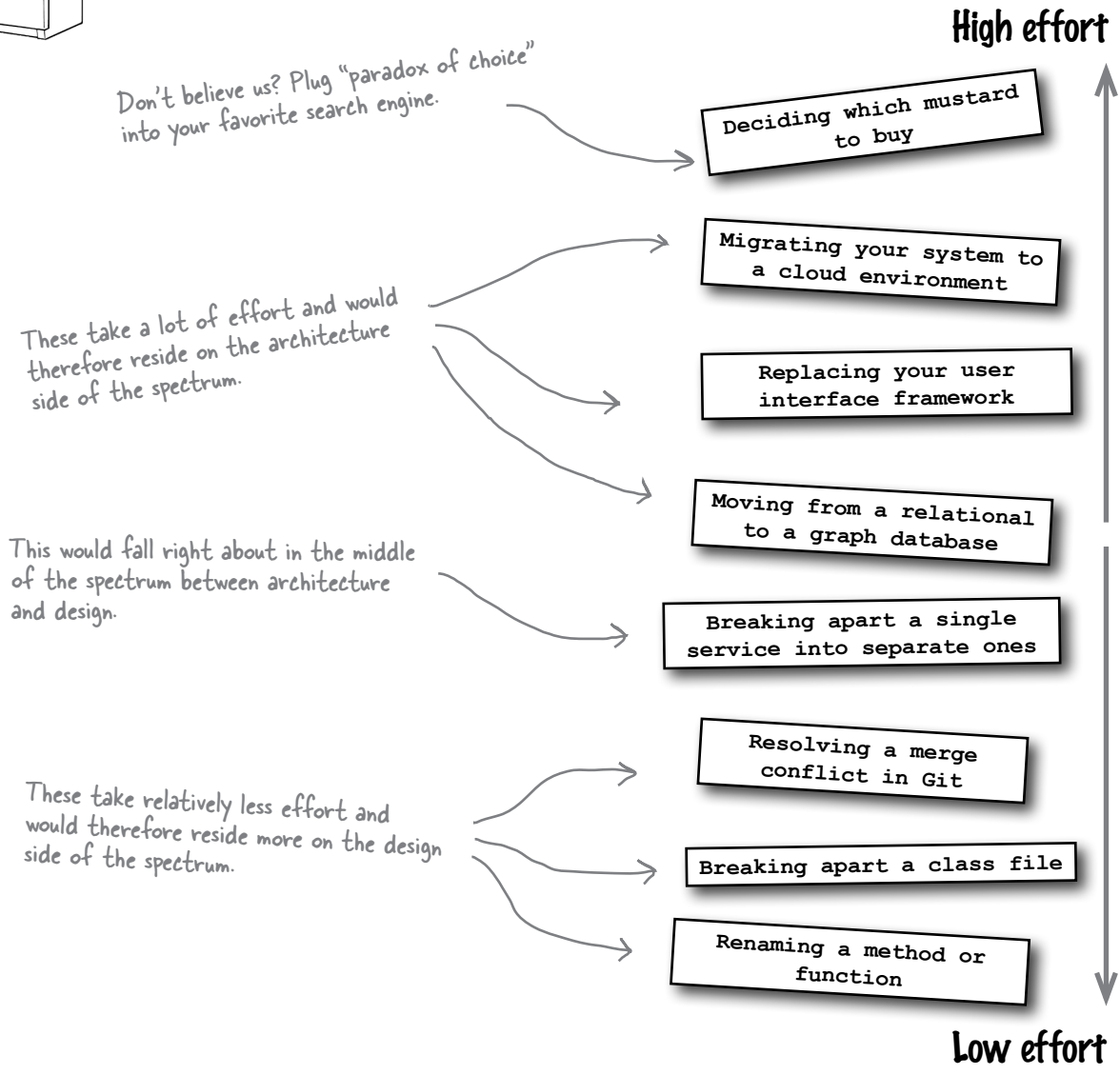




Code Magnets Solution

From page 23

We had all of these magnets from our to-do list arranged from high effort to low effort, and somehow they all fell on the floor and got mixed up. Can you help us put them back in the right order based on the amount of effort it would take to make each change?





Exercise Solution

From page 25

Decisions, decisions, decisions. How can we ever tackle all of these decisions? One thing we think might help is to identify the decisions that involve significant trade-offs, since those will require more thinking and will take longer. Can you help us by identifying which decisions have significant trade-offs and which ones don't?

Significant Tradeoffs?

- | | | |
|---|--|--|
| <input type="checkbox"/> Yes | <input checked="" type="checkbox"/> No | Picking out what clothes to wear to work today |
| <input checked="" type="checkbox"/> Yes | <input type="checkbox"/> No | Choosing to deploy in the cloud or on premisis |
| <input type="checkbox"/> Yes | <input checked="" type="checkbox"/> No | Selecting a user interface framework |
| <input type="checkbox"/> Yes | <input checked="" type="checkbox"/> No | Deciding on the name of a variable in a class file |
| <input type="checkbox"/> Yes | <input checked="" type="checkbox"/> No | Choosing between vanilla and chocolate ice cream |
| <input checked="" type="checkbox"/> Yes | <input type="checkbox"/> No | Deciding which architectural style to use |
| <input checked="" type="checkbox"/> Yes | <input type="checkbox"/> No | Choosing between REST and messaging |
| <input checked="" type="checkbox"/> Yes | <input type="checkbox"/> No | Using full data or only keys for the message payload |
| <input type="checkbox"/> Yes | <input checked="" type="checkbox"/> No | Selecting an XML parsing library |
| <input checked="" type="checkbox"/> Yes | <input type="checkbox"/> No | Deciding whether or not to break apart a service |
| <input checked="" type="checkbox"/> Yes | <input type="checkbox"/> No | Choosing between atomic or distributed transactions |
| <input type="checkbox"/> Yes | <input checked="" type="checkbox"/> No | Deciding whether or not to go out to dinner tonight |

Okay, so maybe this is a difficult decision sometimes.

There are certainly trade-offs here, so this one could go either way.

These can impact scalability, performance, and overall maintainability.

Are you getting hungry yet?

This can impact data integrity and data consistency, but also scalability and performance.

True or False Solution

True False Design is like the structure of a house (walls, roof, layout, and so on), and software architecture is like the furniture and decoration.

This is backwards.

True False Most decisions are purely about architecture or design. Very few exist along a spectrum between architecture and design.

Most decisions lie within the spectrum between architecture and design.

True False The more strategic your decision, the more it's about architecture; the more tactical, the more it's about design.

True False The more effort it takes to implement or change your decision, the more it's about design; the less effort, the more it's about architecture.

This is backwards.

True False *Trade-offs* are the pros and cons of a given decision or task. The more significant the trade-offs become, the more it's about architecture.

Software Architecture Crossword Solution

From page 23

