Analyzing Business Domains

In this chapter, we will learn domain-driven design tools for analyzing a company’s

business domain and its structure: its **core**, **supporting**, and **generic subdomains.**

# What is a Business Domain?

A business domain **defines a company’s main area of activity**. Generally speaking, it’s

the service the company provides to its clients. For example:

• FedEx provides courier delivery.

• Starbucks is best known for its coffee.

A company can operate in multiple business domains. For example, Amazon provides

both retail and cloud computing services. Uber is a rideshare company that also provides food delivery and bicycle-sharing services.

# What is A Subdomain?

To achieve its business **domain’s goals** and targets, a company has to **operate in multiple**

subdomains.

A subdomain is a fine-grained area of business activity.

All of a company’s subdomains form its business domain: the service it provides to its customers.

Implementing a single subdomain is not enough for a company to succeed; it’s just

one building block in the overarching system.

The subdomains have to interact with each other to achieve the company’s goals in its business domain. For example, Starbucks may be most recognized for its coffee, but building a successful coffeehouse chain requires more than just knowing how to make great coffee. You also have to buy or rent real estate at effective locations, hire personnel, and manage finances, among other activities. None of these subdomains on its own will make a profitable company. All of them together are necessary for a company to be able to compete in its business domain(s).

subdomains bear **different strategic/business values**. Domain-driven design distinguishes between three types of subdomains: core, generic, and supporting. Let’s see how they differ **from a company strategy point of view.**

## Core Subdomains

A core subdomainis what a company does differently from its competitors. This may

involve inventing new products or services or reducing costs by optimizing existing processes.

Let’s take Uber as an example. Initially, the company provided a novel form of transportation: ridesharing. As its competitors caught up, Uber found ways to optimize

and evolve its core business: for example, reducing costs by matching riders heading in the same direction.

To maintain a competitive advantage, core subdomains involve inventions, smart optimizations, business knowhow, or other intellectual property.

### Core Subdomains are Naturally Complex

A core subdomain that is simple to implement can only provide a short-lived competitive advantage. Therefore, core subdomains are naturally complex.

There should be high entry barriers for a company’s core business; it should be hard for

competitors to copy or imitate the company’s solution.

### Core Subdomains are the Source of Competitive Advantage and are not Necessarily Technical

Not all business problems are solved through algorithms or other technical solutions. A company’s competitive advantage can come from various sources.

Consider a jewelry maker selling its products online. The online shop is important, but it’s not a core subdomain. The jewelry design is. **The company can use an existing off-the-shelf online shop engine, but it cannot outsource the design of its jewelry**. The design is the reason customers buy the jewelry maker’s products and remember the brand.

### Core Domains vs Core Subdomains

Core subdomains are also called core domains. in the original domain-driven design book, Eric Evans uses “core subdomain” and “core domain” interchangeably.

Although the term “core domain” is used often, I prefer to use “core subdomain” for a number of reasons. First, it is a *subdomain,* and I prefer to avoid confusion with *business domains*. Second, as we will see that it’s not uncommon for subdomains to evolve over time and change their types. For example, a core subdomain can turn into a generic subdomain. Hence, saying that “a *generic* subdomain has evolved into a *core* subdomain” is more straightforward than saying “a generic *subdomain* has evolved into a core *domain.*”(*I think they made a typo here but the point is still there.*)

## Generic Subdomains

*Generic subdomains* are **business activities that all companies are performing in the same way**. **Like core subdomains, generic subdomains are generally complex and**

**hard to implement**. **However, generic subdomains do not provide any competitive**

**edge for the company**.

There is no need for innovation or optimization here: battletested implementations are widely available, and all companies use them. **For example**, most systems need to **authenticate and authorize their users**. Instead of inventing a proprietary authentication mechanism, it makes more sense to use an existing solution. Such a solution is likely to be more reliable and secure since it has already been tested by many other companies that have the same needs.

Going back to the example of a jewelry maker selling its products online, jewelry design is a core subdomain, but the online shop is a generic subdomain. Using the same online retail platform—the same generic solution—as its competitors would not impact the jewelry maker’s competitive advantage.

## Supporting Subdomains

As the name suggests, ***supporting subdomains* support the company’s business**. However, contrary to core subdomains, (*and I add: similar to generic subdomains*)supporting subdomains do not provide any competitive advantage.

For example, consider an online advertising company whose core subdomains include matching ads to visitors, optimizing the ads’ effectiveness, and minimizing the cost of ad space.

However, to achieve success in these areas, **the company needs to catalog its creative materials**. The way the company stores and indexes its physical creative materials, such as banners and landing pages, does not impact its profits. There is nothing to invent or optimize in that area. On the other hand, the creative catalog is essential for implementing the company’s advertising management and serving systems. That makes the content cataloging solution one of the company’s supporting subdomains.

The distinctive characteristic of supporting subdomains is the complexity of the solution’s

business logic. Supporting subdomains are simple. Their business logic resembles mostly data entry screens and ETL (extract, transform, load) operations; that is, the so-called CRUD (create, read, update, and delete) interfaces. These activity areas do not provide any competitive advantage for the company, and therefore do not require high entry barriers.

*I add: generic subdomains are infrastructures that are needed for the business to operate, they can be used across different businesses with little to no customizations and there are ready-to-use solutions for them. supporting subdomains on the other hand are specific to that business, yet they don’t provide the same value as the core subdomains. they are not complex. these are just the things that has to be done yet there is no ready-to-use solution for them.*

# Comparing Subdomains

## Competitive Advantage

Only core subdomains provide a competitive advantage to a company. Core subdomains

are the company’s strategy for differentiating itself from its competitors.

Generic subdomains, by definition, cannot be a source for any competitive advantage. These are generic solutions—the same solutions used by the company and its competitors.

Supporting subdomains have low entry barriers and cannot provide a competitive advantage either. Usually, a company wouldn’t mind its competitors copying its supporting subdomains—this won’t affect its competitiveness in the industry. On the contrary, **strategically the company would prefer its supporting subdomains to be generic, ready-made solutions, thus eliminating the need to design and build their implementation**. We will learn in detail about such cases of supporting subdomains turning into generic subdomains, as well as other possible permutations.

## Complexity

**Supporting subdomains’ business logic is simple. These are basic ETL operations and**

**CRUD interfaces, and the business logic is obvious. Often, it doesn’t go beyond validating inputs or converting data from one structure to another.**

Generic subdomains are much more complicated. There should be a good reason why others have already invested time and effort in solving these problems. These solutions are neither simple nor trivial. Consider, for example, **encryption algorithms or authentication mechanisms.**

From a knowledge availability perspective, **generic subdomains are “known unknowns.”** These are **the things that you know you don’t know**. Furthermore, this knowledge is readily available. You can either use industry-accepted best practices or, if needed, hire a consultant specializing in the area to help design a custom solution.

Core subdomains are complex. They should be as hard for competitors to copy as possible—the company’s profitability depends on it. That’s why **strategically**, companies are looking to solve complex problems as their core subdomains.

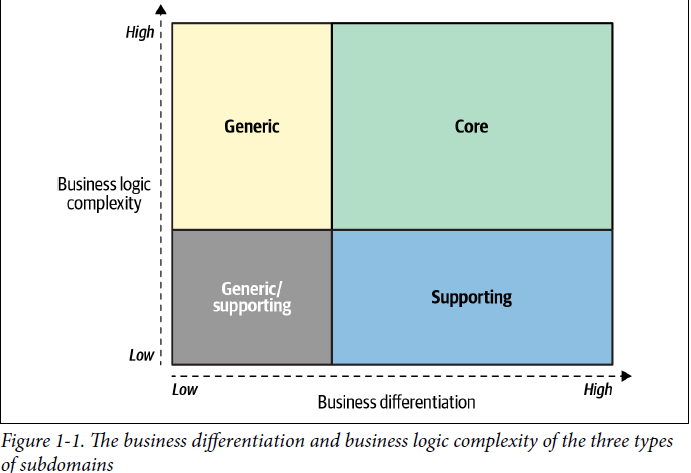
* At times it may be challenging to differentiate **between core and supporting** subdomains. **Complexity is a useful guiding principle**. Ask whether the subdomain in question can be turned into a side business. **Would someone pay for it on its own? If so, this is a core subdomain**.
* Similar reasoning applies for differentiating **supporting and generic subdomains:** **would it be simpler and cheaper to hack your own implementation, rather than integrating an external one**? **If so, this is a supporting subdomain.**

From a more technical perspective, it’s important to identify **the core subdomains whose complexity will affect software design.** As we discussed earlier, a core subdomain is not necessarily related to software. Another useful guiding principle for identifying software-related core subdomains is to evaluate the complexity of the business logic that you will have to model and implement in code. Does the business logic resemble CRUD interfaces for data entry, or do you have to implement **complex algorithms** or business processes orchestrated by **complex business rules and invariants**? In the former case, it’s a sign of a supporting subdomain, while the latter is a typical **core subdomain**.

The chart below represents the interplay between the three types of subdomains

in terms of business differentiation **and** business logic complexity.

**The intersection between the supporting and generic subdomains is a gray area**: **it can go either way.** If a generic solution exists for a supporting subdomain’s functionality, the resultant subdomain type depends on whether it’s simpler and/or cheaper to integrate the generic solution than it is to implement the functionality from scratch.



## Volatility

As mentioned previously, core subdomains can change often. If a problem can be solved on the first attempt, it’s probably not a good competitive advantage—competitors

will catch up fast. Consequently, solutions for core subdomains are emergent.

Different implementations have to be tried out, refined, and optimized. Moreover, the

work on core subdomains is never done. Companies continuously innovate and

evolve core subdomains. The changes come in the form of adding new features or

optimizing existing functionality. Either way, the constant evolution of its core subdomains

is essential for a company to stay ahead of its competitors.

Contrary to the core subdomains, **supporting subdomains do not change often**. They do not provide any competitive advantage for the company, and therefore the **evolution of a supporting subdomain provides a minuscule business value compared to** the same effort invested in a **core subdomain.**

**Despite having existing solutions**, **generic subdomains can change over time**. The

changes can come in the form of **security patches, bug fixes, or entirely new solutions**

**to the generic problems.**

## Solution/implementation Strategy

Core subdomains provide the company its ability to compete with other players in

the industry. That’s a business-critical responsibility, but does it mean that supporting

and generic subdomains are not important? Of course not**. All subdomains are**

**required for the company to work in its business domain**. The subdomains are like

foundational building blocks: **take one away and the whole structure may fall down**.

That said, **we can leverage the inherent properties of the different types of subdomains to choose implementation strategies to implement each type of subdomain in the most efficient manner.**

* **Core subdomains have to be implemented in-house.** They cannot be bought or adopted; that would undermine the notion of competitive advantage, as the company’s competitors would be able to do the same.

**It would also be unwise to outsource the implementation of a core subdomain**. **It is a strategic investment**. Cutting corners on a core subdomain is not only risky in the short term but can have fatal consequences in the long run: for example, unmaintainable codebases that cannot support the company’s goals and objectives.

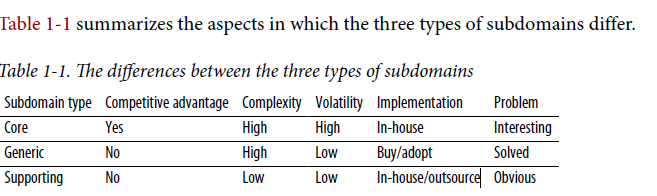
**The organization’s most skilled talent should be assigned to work on its core subdomains**. Furthermore, implementing core subdomains in-house allows the company to make changes and evolve the solution more quickly, and therefore build the competitive advantage in less time.

**Since core subdomains’ requirements are expected to change often and continuously, the solution must be maintainable and easy to evolve. Thus, core subdomains require implementation of the most advanced engineering techniques.**

* Since generic subdomains are hard but already solved problems, **it’s more cost-effective to buy an off-the-shelf product or adopt an open-source solution** than invest time and effort into implementing a generic subdomain in-house.
* Lack of competitive advantage makes it reasonable to avoid implementing supporting subdomains in-house. However, **unlike generic subdomains, no ready-made solutions are available**. So, a company has no choice but to implement supporting subdomains itself. That said, **the simplicity of the business logic and infrequency of changes make it easy to cut corners.**

**Supporting subdomains do not require elaborate design patterns or other advanced engineering techniques.** A rapid application development framework will suffice toimplement the business logic without introducing accidental complexities.

From a staffing perspective**, supporting subdomains** do not require highly skilled technical aptitude and provide a **great opportunity to train up-and-coming talent**. **Save the engineers** on your team who are experienced in tackling complex challenges **for the core subdomains**. Finally, **the simplicity of the business logic makes supporting** **subdomains a good candidate for outsourcing.**



# Identifying Subdomain Boundaries

As we can already see, identifying subdomains and their types can help considerably

in making different design decisions when building software solutions. But how can we identify subdomains and their boundaries?

The subdomains and their types are defined by the company’s business strategy: its

business domains and **how it differentiates itself to compete with other companies in**

**the same field.**

In the vast majority of software projects, in one way or another **the subdomains are already there.** That **doesn’t mean, however, that it is always easy and straightforward to identify their boundaries**. If you ask a CEO for a list of their company’s subdomains, you will probably receive a blank stare. They are not aware of this concept. Therefore, **you’ll have to do the domain analysis yourself to identify and categorize the subdomains at play.**

* A good starting point is the company’s departments and other organizational units.

For example, an online retail shop might include warehouse, customer service, picking, shipping, quality control, and channel management departments, among others.

**These, however, are relatively coarse-grained areas of activity**. Take, for example, the

customer service department. It’s reasonable to assume that it would be a supporting,

or even a generic subdomain, as this function is often outsourced to third-party vendors.

But **is this information enough for us to make sound software design decisions?**

## Distilling subdomains

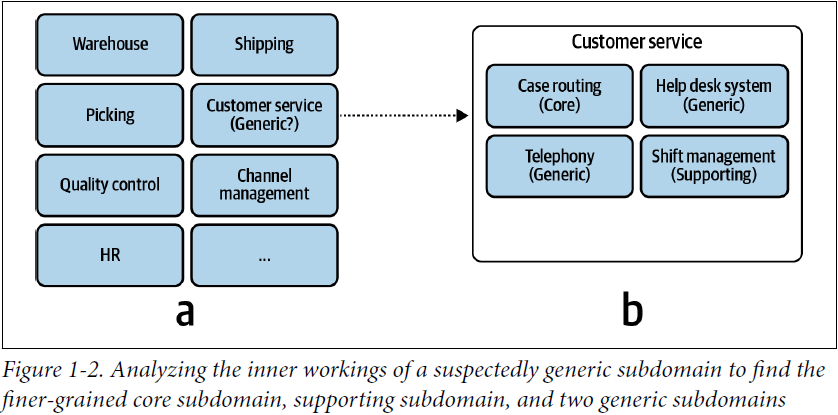
**Coarse-grained subdomains are a good starting point, but the devil is in the details. We have to make sure we are not missing important information hidden in the intricacies of the business function.** Let’s get back to the customer service example

**If we investigate** its **inner workings**, we will see that a **typical customer service department is composed of finer-grained components**, such as

* a help desk system
* shift management and scheduling
* telephone system, and so on.

When viewed as individual subdomains, these activities can be of different types:

* while help desk and telephone systems are generic subdomains
* shift management is a supporting one
* **while a company may develop its ingenious algorithm for routing incidents to agents having success with similar cases in the past. The routing algorithm requires analyzing incoming cases and identifying similarities in past experience—both of which are nontrivial tasks. Since the routing algorithm allows the company to provide a better customer experience than its competitors, the routing algorithm is a core subdomain.** This example is demonstrated in the figure below:



On the other hand, we cannot drill down indefinitely, looking for insights at lower

and lower levels of granularity. When should you stop?

## Subdomains as coherent use cases

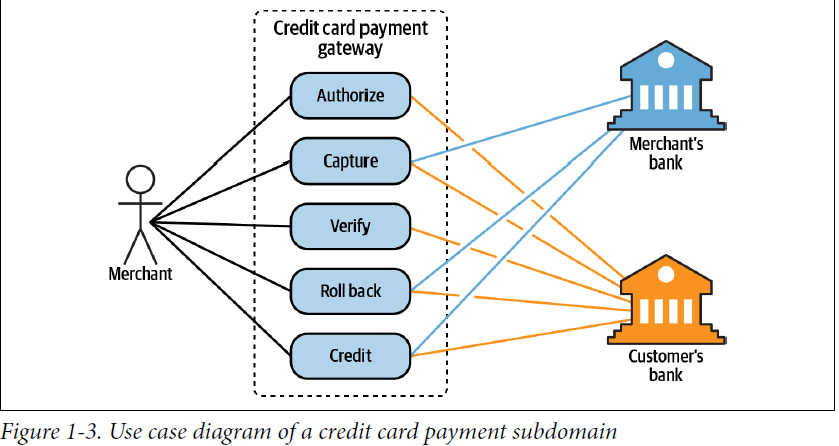
From **a technical perspective**, subdomains resemble **sets of interrelated**, **coherent use**

**cases**. Such sets of use cases usually involve **the same actor, the business entities**, and

**they all manipulate a closely related set of data**.

Consider the use case diagram for a credit card payment gateway shown in

The figure below:



The use cases are tightly bound by the data they are working with and the involved actors. Hence, all of the use cases form the credit card payment subdomain.

We can use the definition of “subdomains as a set of coherent use cases” as a guiding

principle **for when to stop looking for finer-grained subdomains**. These are the most

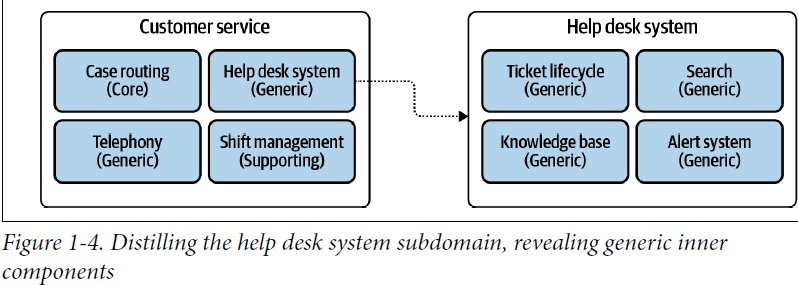
precise boundaries of the subdomains.

Should you always strive to identify such laser-focused subdomain boundaries?

* **It is definitely necessary for core subdomains**. Core subdomains are **the most important**, **volatile**, **and complex**. It’s essential that we distill them as much as possible since that will allow us to extract all generic and supporting functionalities and invest the effort on a much more focused functionality.
* The distillation can be somewhat relaxed for supporting and generic subdomains. If drilling down further doesn’t unveil any new insights that can help you make software design decisions, it can be a good place **to stop**. This can happen, for example, **when all of the finer-grained subdomains are of the same type as the original subdomain.**

*I add : for example, when you see that all the sub-parts of a generic subdomain are themselves generic, they are working tightly together and for example breaking it to parts won’t reveal something like using a separate technology for a specific part that could make your life easier, there is no good in digging further.*

Further distillation of the help desk system subdomain for example, is less useful, as it doesn’t reveal any strategic information, and a coarse-grained, off-the-shelf tool will be used as the solution:



**Another important question to consider when identifying the subdomains is whether we need all of them.**

## Focus on the Essentials

**Subdomains are a tool that alleviates the process of making software design decisions**.

All organizations likely have quite a few business functionalities that drive their competitive

advantage but have nothing to do with software. The jewelry maker we discussed

earlier in this chapter is but one example.

When looking for subdomains, it’s important to identify business functions that are

not related to software, acknowledge them as such, and focus on aspects of the business

that are relevant to the software system you are working on.

# Domain Analysis Examples

Let’s apply the notion of subdomains in practice and use it for making a number of **strategic design decisions:**

## Go and Read the Examples in the Book, Learning Domain Driven Design.

# Who are Domain Experts?

Domain experts are subject matter experts who know all the intricacies of the business that we are going to model and implement in code. In other words, domain experts are **knowledge authorities in the software’s business domain.**

**The domain experts are neither the analysts** **gathering the requirements nor the engineers designing the system**. Domain experts represent the business. **They are the people who identified the business problem in the first place** **and from whom all business knowledge originates.** Systems analysts and engineers are transforming their mental models of the business domain into software requirements and source code.

**As a rule of thumb, domain experts are either the people coming up with requirements or the software’s end users.** **The software is supposed to solve their problems.**

# Conclusion

A **business domain** is the main area in which a business operates and the service it provides to its clients.

**Subdomains** are building blocks that are required for the business to succeed in its domain.

**Core subdomains** ------------------------------> the interesting problems

**Generic subdomains**---------------------------> the solved problems

**Supporting subdomains**-----------------------> the problems with obvious solution

**Domain Experts** are the business’s subject matter experts with deep knowledge of the business domain or in one or more subdomain.