Monolithic vs Microservices software architecture

Choosing the right design for your app development

Willem L. Middelkoop
Mar. 3, 2020

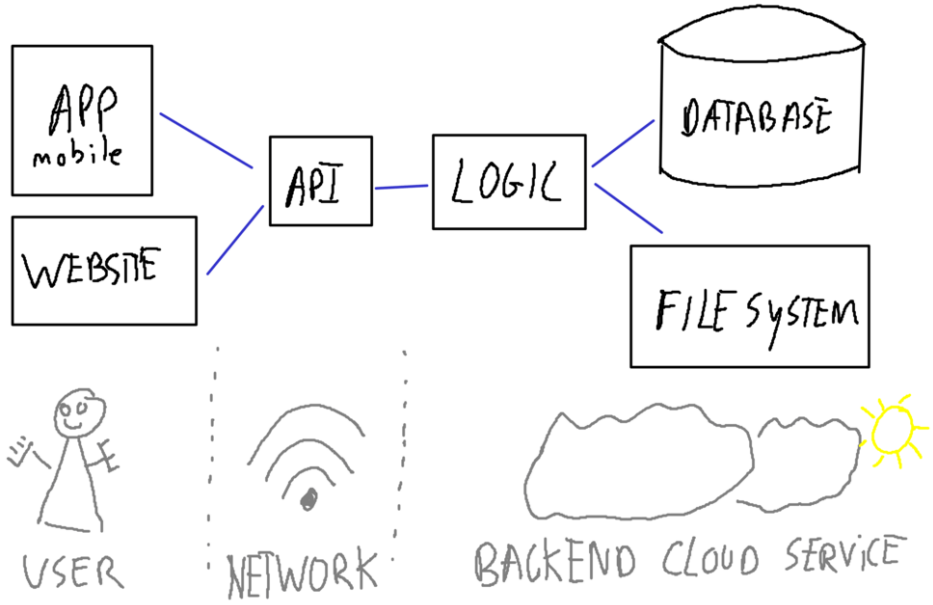
This week I flew to Gothenburg to meet people from a large international shipping company, talking about the development of enterprise level software. During the meeting there were various experts in the room, one of them asked me on choosing the right software architecture (for big, complex, enterprise level apps). A very good question, well worthy for a blog post.

Visiting a large international shipping company to talk about enterprise software

Software architecture

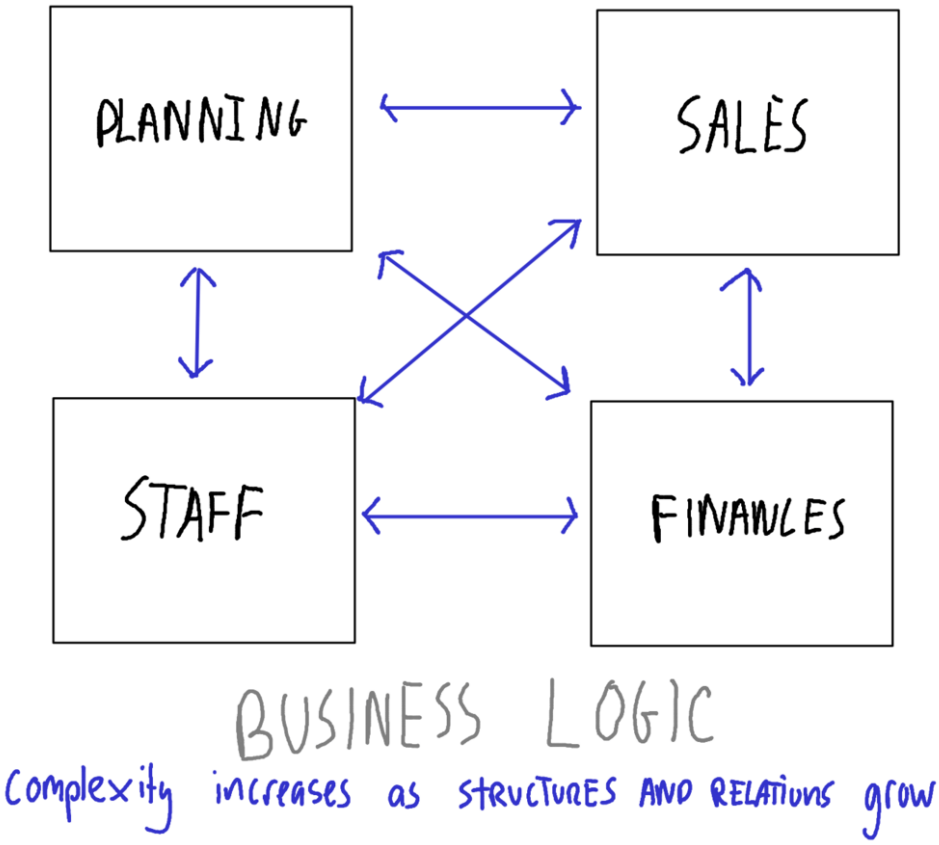
When you design apps you should not only think about the mere look and feel, good design encompasses how everything works together, too. Software architecture refers to the fundamental structures of a software system, each structure comprises elements, relations among them, and properties of both elements and their relations.

It is analogous to the architecture of a building. Once an architecture is defined and implemented, it becomes costly to change it afterwards. It is therefore a good idea to choose the right software architecture *before* you start building.



Simple software architecture example - different structures composing a system with different relations between them

For simple apps the software architecture could be obvious, but as your application grows in size and functionality, the number of structures in a system can grow rapidly. This can happen over time, as your customer's needs grows. The more structures, the more relations between them, the complexer it gets.



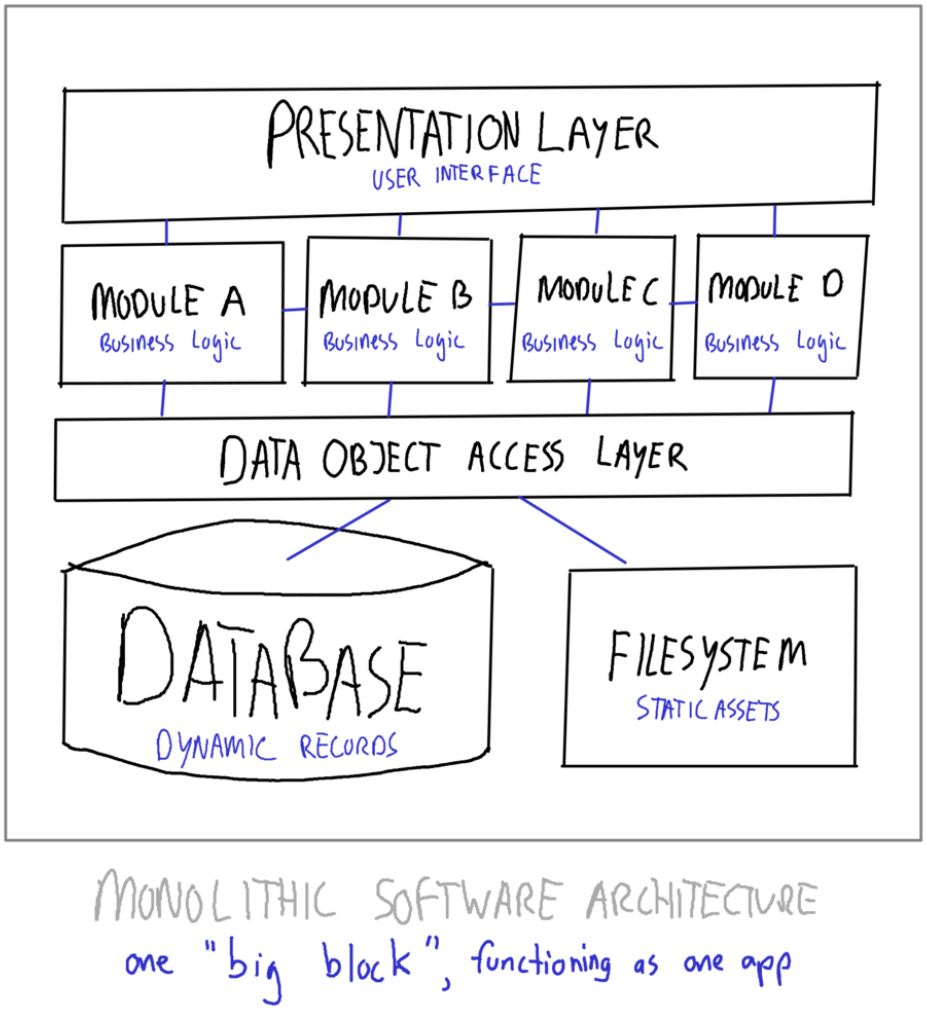
Increased complexity as the number of structures (and their relations) grows

Dealing with complex software design

If you expect your software to handle a lot of different things, chances are you're going to deal with complexity. From an architectural point of view there are different approaches in dealing with this software complexity.

Monolithic Architecture

Best described as 'one big block' is the monolithic architecture. Here, everything your app does, is part of the same code base.



Monolithic software architecture

In a "monolithic architecture" the software is often layered, where each layer consists of different types of components:

- **presentation layer**: responsible for the user and/or application programming interface (UI / API).
- **business logic**: this is where actual business rules are defined. It is the core of the application that is very closely matched to the actual business processes.
- **data object access layer**: providing a common interface between the running application and persistent backend stores.
- **database and filesystem**: this is where all the data and static assets (files, images, etc), are stored.

The bigger the application becomes, the larger each layer becomes. This makes it harder to scale a monolithic architecture, at a certain point things become simply too big and complex.

Every time you make a change to the application, the entire code base is affected. For (very) large applications this makes it harder to maintain the software, as it requires a full understanding of the entire application.

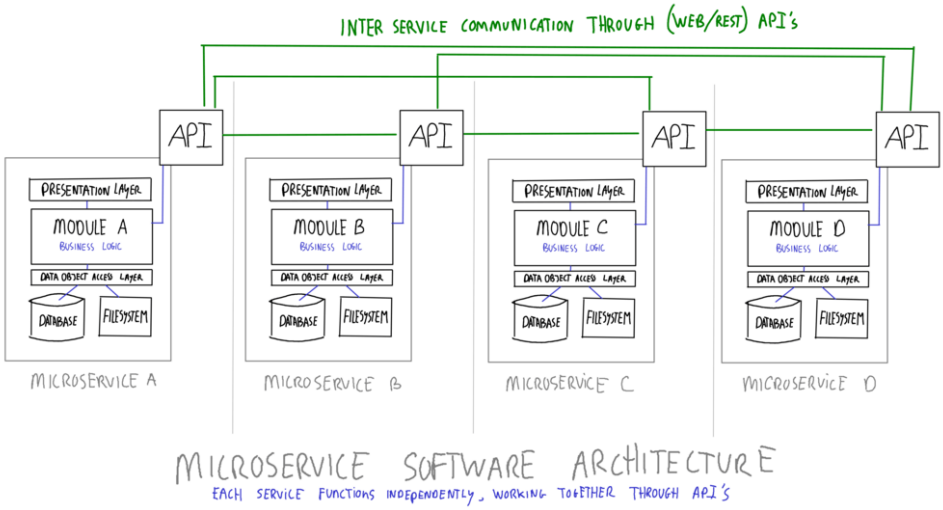
As the entire application is interdependent, problems in one part of the application often significantly affect the entire system. Communication between different parts of

the system is done internally, at code level. This means that the different parts must be programmed in the same (or compatible) programming language. This is a barrier to adopt new technologies (in the future) as you cannot easily change the programming language of one part of the system.

Microservices architecture

Instead of building one big monolithic application you can split up your application in smaller interconnected services. Each microservice is a small application of its own, complete with its own (but smaller + simpler) architecture, business logic and data layers.

The different services communicate with each other using API's that are defined independently of the used technology, often in a common "API structure" such as REST, JSON, Redis and XML. This allows the combination of microservices to function as "one system".



Microservices Software Architecture

By decomposing the 'big block' application into smaller services, it becomes easier to understand different parts of the system. This makes it easier to maintain the system as it allows each microservice to be developed independently. It also reduces the barrier to adopt new technologies as you are free to choose whatever technology is best suited for a given service. One part of the system can be programmed in a different language than other parts. This makes it possible for different teams (with different expertise) to work together.

The art and skill of getting the microservices architecture right, come from the ability to correctly define a microservice. Too broad or too granular can destroy the architecture. You can use business processes, hierarchical or domain separation to define each microservice.

The challenge with the microservice architecture is that it adds complexity to the entire system by the fact that it requires a distributed communication mechanism. The different API's that connect to each other can be tricky to test and debug. It becomes more difficult to implement changes that span multiple services.

Defining, designing and implementing the different interservice API's require an agreement on the exchanged data, this can be hard to achieve when you intend to share extended and non-universal data structures.

Monitoring and deploying a microservice architecture can be complex when services grow in number. There is simply more to manage, requiring more planning and coordination for each of the services.

Choosing the right architecture

My late dad once told me that you can "earn the most by the things you don't do". Building complex applications is inherently difficult, the best thing you can do is to try to make things simpler by simply not doing everything. Focus on the things that truly matter, do those things right.

If your application is simple, lightweight, a monolithic architecture is the way to go. Development, deployment, maintenance, everything is easier when things are small.

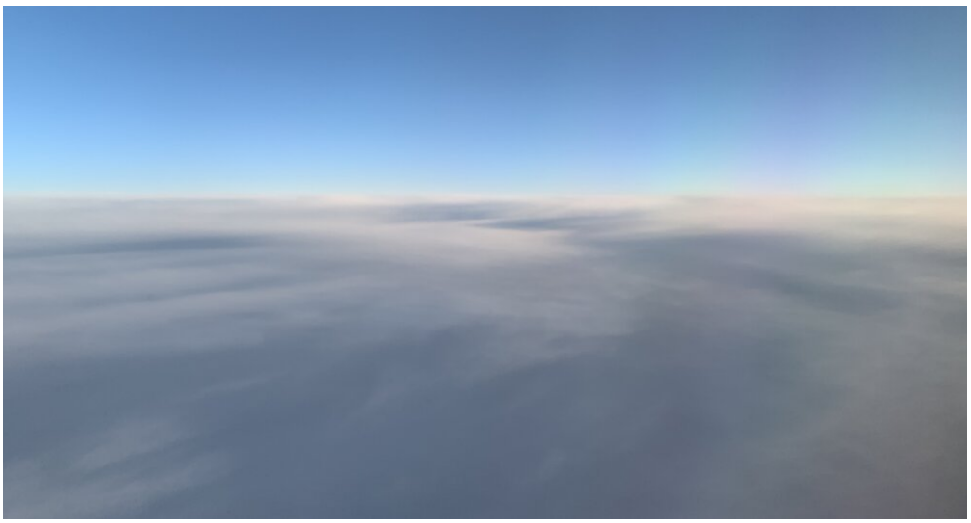
If you cannot escape complexity, then you should definitely consider the microservice architecture. Ask yourself the following questions:

- Are you able to define a "common language" between your microservices?
- Are there clearly distinct business processes?
- How significantly are these business processes interdependent?
- How do you expect your application to evolve over time, do you expect it to grow or shrink?
- How many different programming technologies do you plan to use? Do you expect this to change in the future?
- Who's going to build and maintain the software, do they have experience managing such an architecture?

Conclusion

There is no "silver bullet" answer to the question what software architecture is right. Both approaches have their pros and cons.

The best thing you can do is to make a conscious - well considered - decision and double down on it!



"The sky is the limit" if you master the architecture you have chosen (just don't change it during flight!)