ANALYSIS OF APPROACHES FOR EFFECTIVE APPLICATIONS DEVELOPMENT IN MONOLITH, SERVICE-ORIENTED AND MICROSERVICE ARCHITECTURE

Yurii Milovidov

Abstract: *The paper presents the features of the using the microservice architecture in the development process. The advantages of this approach are illustrated in comparison with the traditional monolithic approach. The connection between the use of the microservice architecture and the ability of the team to work within agile development methodologies is shown. Creating a new product is often a risk. And choosing the right architecture is an essential step toward success. If you're considering between a monolithic, service-oriented and microservice architecture, this paper will help you make the right choice.*

Key words: *Service Oriented Architecture, microservices, microservice architecture, efficient development, agile methodologies, Quality of Service.*

ITHEA keywords: *D.2 Software engineering, D.2.1 Requirement/Spesification, D.2.13 Reusable Software.*

Introduction

Software architecture modeling is an important part of the development process. The choice of a specific architectural solution depends on many factors, in particular, the purpose of the system being developed. Despite the variety of approaches to the architecture of applications, in the framework of modern projects that need easy scalability, lately, microservices have been preferred [Shadija D. Rezai M. Hill R.].

The term Microservice Architecture describes a software development technology in which an application is a collection of loosely coupled services. Services are built around business logic and are independently deployable. Each service works in its own process and interacts with other services using lightweight mechanisms, such as, for example, HTTP. Moreover, the development of each service can be carried out independently of the others. This allows you to organize the development process flexibly and distributed.

The main goal of this paper is to determine and describe the conditions necessary for the implementation of effective development based on microservice architecture.

Theoretical foundations

Service-Oriented Architecture (SOA) is a software architecture where functionality is grouped around business processes and packaged as interoperable services. SOA also describes IT infrastructure which allows different applications to exchange data with one another by passing data from one service to another as they participate in business processes. The aim is a loose coupling of services with operating systems, programming languages and other technologies which underlie applications. SOA separates functions into distinct units, or services, which are made accessible over a network in order that they can be combined and reused in the production of business applications.

The microservices approach has become a trend in recent years. Microservices have become the software architecture of choice for business applications. Initially originated at Netflix and Amazon, they result from the need to partition, both, software development teams and executing components, to, respectively, foster agile development and horizontal scalability.

The concept of monolithic software lies in different components of an application being combined into a single program on a single platform. Usually, a monolithic app consists of a database, client-side user interface, and server-side application. All the software's parts are unified and all its functions are managed in one place. A monolithic architecture is comfortable for small teams. Components of monolithic software are interconnected and interdependent, which helps the software be self-contained. This architecture is a traditional solution for building applications.

Related works

Interoperability Constraints in Service Selection Algorithms [Kaczmarek P.]

Tasks: Existing methods and algorithms of service selection focus on the formal model of the complex service refraining from interoperability issues that may affect the integration process as services are deployed in heterogeneous runtime environments. Service integration in SOA requires resolution of interoperability issues.

Approaches for solving task: Alternative services realizing the same functionality differ in non-functional, Quality of Service (QoS) attributes such as performance, reliability and price. It is required that services with optimal QoS values are selected, while constraints are satisfied. Considering the problem, the author proposes a methodology that includes interoperability analysis in service selection algorithms. The data structures of service selection algorithms are extended with dedicated constraints that represent interoperability between services.

If QoS constraints are removed from the selection problem, a simplified model is created, in which services are selected considering solely the utility function. In this paper, interoperability is analyzed on the communication protocol level, which corresponds to Level 1 (Connected) in the LISI metric or Level 2 (Data/Object) in the LCI metric. Was assumed that if services are interoperable on that level, it is possible to exchange data between them in the context of CS construction.

The proposed extensions of existing models with interoperability constraints enable a possibly straightforward application of existing selection algorithms.

As a part of the research, was implemented the Work-flow Integrator system that enables selection of services during composite service development. The aim of the system is to support developers in the selection process depending on given requirements on utility function, QoS attributes, and interoperability constraints.

Were used workflow applications as a concrete implementation of the composite service concept, which enables us to leverage existing solutions in service description and invocation.

SOAQM: Quality model for SOA applications based on ISO 25010 [Franca J. Soares M.]

This paper addresses one gap in researches about SOA. There is a lack of quality models specific for SOA based on most recent ISO 25010. In this paper authors propose a quality model for SOA applications based on quality attributes proposed in ISO 25010. ISO 25010 is currently the most complete version of quality models of ISO. In order to make a proper evaluation of software, relevant quality characteristics of a software product have been proposed in many quality models, including ISO standards. The task of the

paper is to analyze which important contributions were aggregated into the new ISO 25010 regarding SOA applications. In addition, this analysis aims to determine if limitations perceived in ISO 9126 were solved in the most recent ISO 25010. Another proposal of this paper is to investigate the applicability of ISO 25010 to SOA applications.

Approaches for solving task:

All characteristics and sub-characteristics proposed by ISO 25010 were analyzed regarding the real applicability to SOA. Accordingly, authors establish for each characteristics the degree of applicability to SOA. The next step of this analysis is to define how these characteristics can be adapted to the SOA context.

Was proposed a degree of importance that defines which characteristics are more relevant during the SOA application development process. Seven volunteers answered a questionnaire to define the degree of importance. The answer varies from 1 to 5 following the Likert Scale. Was created a Table that presents the results of the analysis of ISO 25010 characteristics and their applicability to SOA projects. First two columns show characteristics and sub-characteristics proposed by ISO 25010. Third column presents the degree of importance of each sub-characteristic with relation to the SOA context. Highlighted sub-characteristics in green are important or very important quality attributes for SOA applications. Sub-characteristics highlighted in yellow are not so important. Quality sub-characteristics considered less important or even irrelevant to SOA applications and therefore were highlighted in red. Finally, the last column presents the likely reasons that justifies the results.

All quality attributes proposed by ISO 25010were analyzed in this research from the SOA perspective using both experts opinion and literature review. As a result, most of quality attributes proposed by ISO 25010 may be applicable for SOA. On the other hand, ISO 25010 has a generic nature and therefore some quality attributes proposed do not apply to SOA domain.

Service Consumer Framework - Managing Service Evolution from a Consumer [Feuerlicht G. Tran H.]

In this paper, was described initial proposal for Service Consumer Framework that attempts to address this problem by providing resilience to changes in external services as these services are evolved or become temporarily

unavailable. Service evolution has been the subject of recent research interest, however the focus of these efforts has been mainly on developing methodologies and tools that help service providers to manage service evolution. There is a pressing need to develop consumer-side methodologies and tools to address these issues from a service consumer perspective.

Approaches for solving task

Was described a proposal for a Service Consumer Framework (SCF) that attempts to address this problem by providing resilience for client applications to changes in external services as these are evolved or become temporarily unavailable. The SCF framework uses a combination of service adaptors and a service router to protect client applications from external changes. Evolution of services is supported by using service adaptors that transform service request and response messages between internal and external services. Service router determines which external services are evoked at runtime, based on their availability and pre-defined processing rules stored in the Service Repository.

The SFC Framework is illustrated in Figure 1 and consists of three layers: Process Layer, Adaptor Layer and Service Layer. The Service Layer incorporates both internal and external services and defines their native interfaces.

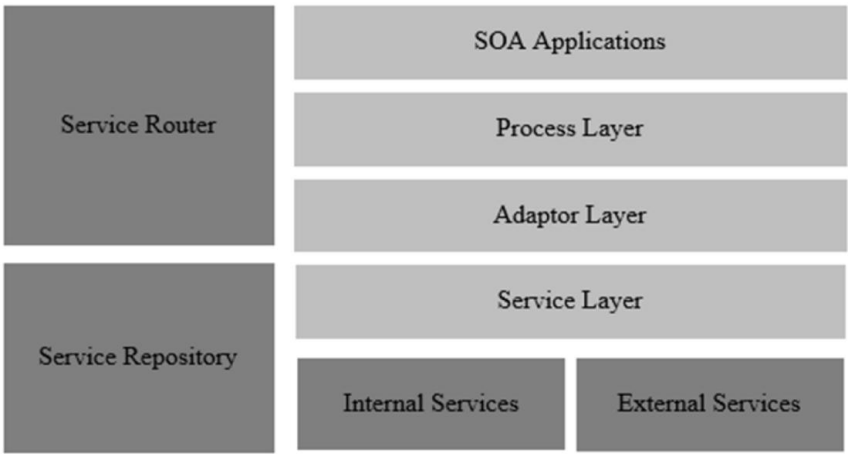

Figure 1. Service Consumer Framework

The Adaptor Layer contains adaptors that translate requests between the native services (e.g. PayPal payment service) and the corresponding internal services. The Process Layer defines processes that are implemented using the service router and processing rules stored in the service repository.Service router determines at runtime, which services are evoked based on their availability and

pre-defined processing rules. Service repository maintains information about services allowing substitution of services with equivalent functionality to avoid service interruptions.

From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts [Nunes L., Santos N., Rito Silva A.]

Currently, there is a large number of monolith applications that are being migrated to a microservices architecture. This article proposes the identification of business applications transactional contexts for the design of microservices. Therefore, the emphasis is to drive the aggregation of domain entities by the transactional contexts where they are executed, instead of by their domain structural inter-relationships. Additionally, authors propose a complete workflow for the identification of microservices together with a set of tools that assist the developers on this process.

Task

The purpose of this paper is to, based on a comparative analysis of different architectural approaches, determine and describe the conditions necessary for effective development on the basis of microservice architecture.

Research methods

This section describes the advantages of microservice architecture compared to monolithic architectures and describes the proposed principles for organizing effective development in the project team developing applications in microservice architecture.

Monolithic architecture

A monolithic architecture is comfortable for small teams. Components of monolithic software are interconnected and interdependent, which helps the software be self-contained. This architecture is a traditional solution for building applications, but some developers find it outdated.

Pros of a monolithic architecture:

- Simpler development and deployment;
- Fewer cross-cutting concerns;
- Better performance.

Cons of a monolithic architecture:

- Codebase gets cumbersome over time;
- Difficult to adopt new technologies;
- Limited agility.

SOA

A service-oriented architecture (SOA) is a software architecture style that refers to an application composed of discrete and loosely coupled software agents that perform a required function. SOA has two main roles: a service provider and a service consumer. Both of these roles can be played by a software agent.

Pros of SOA

- Reusability of services;
- Better maintainability;
- Higher reliability;
- Parallel development.

Cons of SOA

- Complex management;
- High investment costs;
- Extra overload.

Microservice architecture

Microservice is a type of service-oriented software architecture that focuses on building a series of autonomous components that make up an app. Unlike monolithic apps built as a single indivisible unit, microservice apps consist of multiple independent components that are glued together with APIs.

The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

To explai the microservice style it's useful to compare it to the monolithic style: a monolithic application built as a single unit. Enterprise Applications are often built in three main parts: a client-side user interface (consisting of HTML pages and javascript running in a browser on the user's machine) a database

(consisting of many tables inserted into a common, and usually relational, database management system), and a server-side application. The server-side application will handle HTTP requests, execute domain logic, retrieve and update data from the database, and select and populate HTML views to be sent to the browser. This server-side application is a monolith - a single logical executable. Any changes to the system involve building and deploying a new version of the server-side application. The price of microservices pays off by reducing the costs of reducing team productivity while complicating the system. This is shown in Figure 2, which is presented in article [Fowler M.].

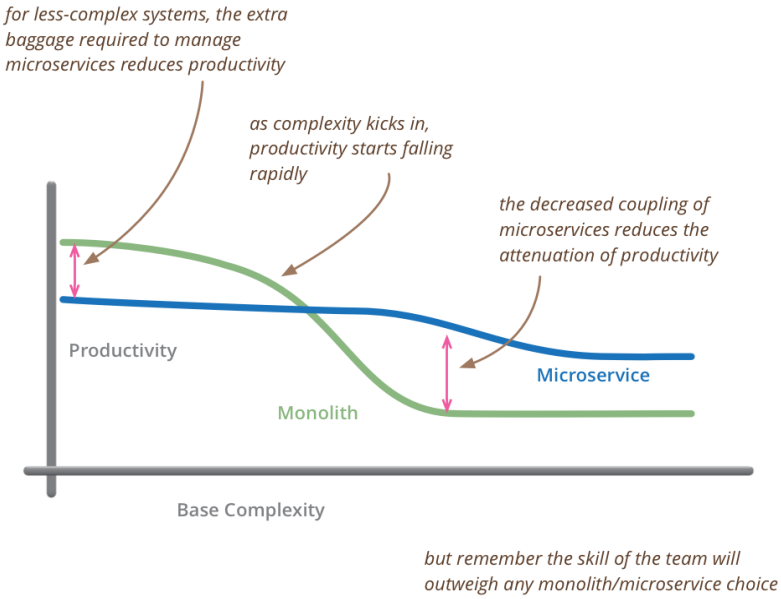

Figure 2. Dependence of team productivity from base complexity.

Pros of microservices

- Easy to develop, test, and deploy. The biggest advantage of microservices over other architectures is that small single services can be built, tested, and deployed independently. Since a deployment unit is small, it facilitates and speeds up development and release. Besides, the release of one unit isn't limited by the release of another unit that isn't finished. And the last plus here is that the risks of deployment are reduced as developers deploy parts of the software, not the whole app.
- Increased agility. With microservices, several teams can work on their services independently and quickly. Each individual part of an application can be built independently due to the decoupling of microservice

components. For example, you may have a team of 100 people working on the whole app (like in the monolithic approach), or you can have 10 teams of 10 people developing different services for the app. Increased agility allows developers to update system components without bringing down the application.

- Ability to scale horizontally. Vertical scaling (running the same software but on bigger machines) can be limited by the capacity of each service. But horizontal scaling (creating more services in the same pool) isn't limited and can run dynamically with microservices. Furthermore, horizontal scaling can be completely automated.

Cons of microservices

- Complexity. The biggest disadvantage of microservices lies in their complexity. Splitting an application into independent microservices entails more artifacts to manage. This type of architecture requires careful planning, enormous effort, team resources, and skills.
- Security concerns. In a microservices application, each functionality that communicates externally via an API increases the chance of attacks. These attacks can happen only if proper security measurements aren't implemented when building an app.
- Different programming languages. The ability to choose different programming languages is two sides of the same coin. Using different languages make deployment more difficult. In addition, it's harder to switch programmers between development phases when each service is written in a different language.

When using microservices, the role of the project manager comes to the fore, since many different teams work on different parts of the application. In order to synchronize the work of microcommands and avoid problems, you should hire competent project managers.

Software architect Simon Brown suggested what a flexible software architecture would look like [14]. His description of flexible architecture fits quite well on microservice architecture. In his opinion, an architecture style built using small loosely coupled components (services) that interact with each other to respond to a user's request will help provide flexibility. Services can be modified and tested in isolation or even torn out and replaced depending on changing requirements, new components can be added and scaled, if necessary.

According to Agile Manifesto, successful and efficient teams make the necessary decisions together, and the recommended team size is at least 3 and no more than 9 people.

In this way, Agile provides support for effective collaborative development in the microservice architecture.

Leading technology organizations such as Amazon, Netflix, and Apple use the microservice architecture to provide the flexibility to quickly adapt and respond to their customers' requests.

The figure 3 shows the differences between the three different architectures that were discussed above.

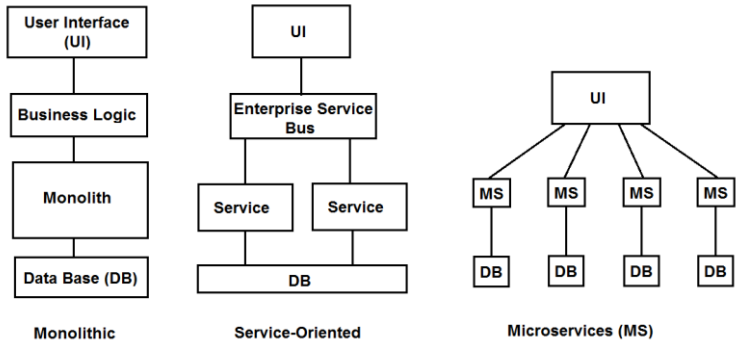

Figure 3. The differences between the three different architectures.

Monolithic applications consist of interdependent, indivisible units and feature very low development speed. SOA is broken into smaller, moderately coupled services, and features slow development. Microservices are very small, loosely coupled independent services and feature rapid continuous development.

Results

The essence of three different architectural approaches is shown below.

The monolithic model isn't outdated, and it still works great in some cases. Some giant companies like Etsy stay monolithic despite today's popularity of microservices. Monolithic software architecture can be beneficial if your team is at the founding stage, you're building an unproven product, and you have no experience with microservices. Monolithic is perfect for startups that need to get a product up and running as soon as possible.

The SOA approach is best suited for complex enterprise systems such as those for banks. A banking system is extremely hard to break into microservices. But a monolithic approach also isn't good for a banking system as one part could hurt the whole app. The best solution is to use the SOA approach and organize complex apps into isolated independent services.

Microservices are good, but not for all types of apps. This pattern works great for evolving applications and complex systems. Choose a microservices architecture when you have multiple experienced teams and when the app is complex enough to break it into services. When an application is large and needs to be flexible and scalable, microservices are beneficial.

Conclusions

By comparing three different architectural approaches, we can draw the following conclusion:

Choosing the right architecture is a daunting task.

When developing a software product, you need to seriously approach the choice of architectural solution. Microservices seem to be a suitable choice for the implementation of large systems involving long-term development. However, microservice architecture carries a lot of risks. In this case, the success of the project depends largely on the availability of an experienced system architect and project manager.

A team using microservice architecture should be formed on the basis of business opportunities. One of the prerequisites for the successful implementation of the microservice architecture project is the competence of the team.

Bibliography

[Kaczmarek P. 2012] Interoperability Constraints in Service Selection Algorithms, Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE-2012), pp 23-32 ISBN: 978-989-8565-13-6 – Access mode: https://scitepress.org/papers/2012/39805/39805.pdf

[Feuerlicht G. Tran H. 2014] Service Consumer Framework - Managing Service Evolution from a Consumer Perspective, Proceedings of the 16th International Conference on Enterprise Information Systems (ICEIS-2014),

pp 665-672 ISBN: 978-989-758-028-4 – Access mode: https://www.scitepress.org/papers/2014/49766/49766.pdf

[Franca J. Soares M. 2015] SOAQM: Quality model for SOA applications based on ISO 25010, Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS-2015), pp 60-70 ISBN: 978-989-758-097-0 – Access mode: https://www.researchgate.net/publication/276202563_SOAQM_Quality_model_for_SOA_applications_based_on_ISO_25010

[Nunes L., Santos N., Rito Silva A. 2019] From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts.: Software Architecture ECSA 2019 pp 37-52 ISBN: 978-3-030-29982-8 – Access mode: https://link.springer.com/chapter/10.1007/978-3-030-29983-5_3

[Shadija D. Rezai M. Hill R 2017] Towards an understanding of microservices, 23rd International Conference on Automation and Computing (ICAC) ISBN: 978-1-5090-5040-6 – Access mode: https://ieeexplore.ieee.org/abstract/document/8082018

[Fowler M. 2015] Microservice Premium, – Access mode: https://martinfowler.com/bliki/MicroservicePremium.html

[Brown S. 2013] Coding the Architecture blog. , – Access mode: http://www.codingthearchitecture.com/2013/09/03/what_is_agile_software_architecture.html

Agile Manifesto – Access mode: http://agilemanifesto.org/

Authors' Information

***Yurii Milovidov**, National University of Life and Enironmental Sciences of Ukraine, Kyiv, scientific Department of Programming Technologies. Senior Lecturer.*

E-mail: milovidov.email.ua

Programming technologies, web-design, Internet – technologies.