International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Architectural Patterns for Migrating WCF-Based Systems to RESTful Microservices on .NET

Venkatesh Muniyandi

Submitted: 01/07/2023 **Revised:** 07/08/2023 **Accepted:** 15/08/2023

Abstract: Migrating legacy Windows Communication Foundation (WCF) systems to RESTful microservices on the .NET platform presents significant architectural and technical challenges. The paper compares the issues faced by WCF with those addressed by design principles of microservices. Our approach recommends specific architectural designs suitable for incremental migration, with new approaches for turning service contracts and simulating session states from old systems. Such patterns are fully available in .NET through the .NET implementation framework that leverages ASP.NET Core and gRPC. Many industry case studies support the conclusion that scaling, maintaining and deploying systems are now less challenging and more agile. Our conclusions give useful directions to teams striving to update traditional software whilst avoiding risks and major disruptions.

Keywords: *WCF migration, RESTful micro services, .NET, architectural patterns, legacy modernization*

1. Introduction

Over the last few years, more enterprises have made cloud computing a central part of their IT systems, allowing them to move old applications to cloud infrastructure. In a number of industries, WCF-based applications are still important because they are strong and fit nicely into old enterprise systems (Kratzke & Quint, 2017). But since cloud-native and microservices are becoming more popular, lots of organizations are required to move their WCF-based applications to solutions that scale and work better. Organizations need to make this move to take advantage of modern cloud services from Microsoft Azure, AWS and Google Cloud, all of which work well with microservices architecture (Silva et al., 2023). That is why there is now a strong effort to bring legacy systems up to date and match them with cloud-native design, as these approaches provide flexibility, reliability and simple scalability.

Moving WCF-based systems to RESTful microservices on the .NET platform introduces many different challenges for both architecture and technical aspects. Mainly developed for SOAs, WCF relies on a close relationship between services, SOAP protocol, session information management and messages sent at the same time (Balalaie et al., 2015). By comparison, RESTful microservices are stateless, lightweight and prefer asynchronous interactions, so their use requires major rethinking of system design and communication methods. In migrating, infrastructure can suffer from performance delays, the inability to support future growth and challenges blending with other systems which all should be resolved during the architectural transition.

Some strategies exist for moving traditional applications to microservices (Pahl & Jamshidi, 2016), but only a small number handle the unique problems related to migrating applications developed using WCF to RESTful services. It is challenging to integrate service contracts used by WCF with the main ideas of RESTful principles, especially because WCF offers custom state and security features. Current methods for migrating systems commonly ignore the requirements of WCF applications for service size, converting messages and migration done in steps (Patel & Sharma, 2023). This work offers new patterns that address these challenges and introduces a method for making choices on the most efficient way to move certain parts of a WCF system. It both deals with technical problems and offers a method for picking the right design patterns based on a

Affiliation: Independent Researcher, Country: Houston, USA, Role: Technical Architect
Email: venky.m@gmail.com

system's needs, business sense and the risk involved.

The purpose of this paper is to introduce and analyze new patterns to guide the move from WCF to RESTful microservices in .NET. It covers locating main issues during migration, finding useful ways to enhance service contracts, deal with legacy session management and secure microservices and checking the solutions by running real-life case studies. Furthermore, the paper hopes to give practitioners a tool to help them find the optimal migration approach for their particular legacy WCF applications. This paper introduces: (1) special methods for migrating from WCF to REST, (2) their assessment in actual WCF-to-REST case studies and (3) a complete framework for making migration choices.

In order to verify the proposed approach and movements, this study uses a mix of interviewing experts and analyzing the outcomes of case examples. Yin (1994) shows that case study research can help us examine how WCF legacy systems are moved to microservices by observing numerous real-life scenarios. By reviewing responses from developers and architects and by looking at data on how the systems perform, scale and work, the study provides a strong assessment of the suggested approaches.

2. Related Work

For years, WCF has been an important technology for building large-scale applications because it allows for building distributed systems easily. WCF systems use SOA to allow components to talk to one another by following the rules in service contracts that outline actions and formats. WCF is mainly known for allowing various communication methods such as SOAP and REST and for handling diverse service configurations (Chakravarthy, 2013). Still, WCF makes it challenging by creating tight dependences, requiring session handling and sticking to SOAP which slows the adoption of these applications in modern cloud-native structures. Because WCF is difficult to change and can be complex, microservices setups are challenging to implement. They should be handled correctly during migration, as they shape the design of the new system.

Microservices make it easier to build systems that are spread out across different parts, unlike WCF. Within .NET, RESTful microservices are a good choice because they work without saving state and use asynchronous communication helps manage large loads of traffic (Newman, 2015). After the introduction of .NET Core and ASP.NET Core, microservices have gained wider adoption in the .NET world. As a result of these technologies, developers can take apart a single monolithic application into independent services that can be added or removed as needed. Switching to RESTful microservices breaks away from WCF mainly in the ways it manages data, handles security and connects different services. As they move away from WCF towards microservices, businesses must make sure the new microservice structure works with their previous systems. Because of new .NET features, gRPC and better container support, it is now simpler to replace old systems with modern and flexible solutions (Johnson & Lee, 2023).

Many methods for updating legacy WCF applications to new architectures have been presented. Jamshidi et al. (2016) recommend using several architectural patterns which help with the transition while highlighting how working in different phases helps to protect the service and lower risks. Most often, teams practice Strangler Fig Pattern in which older code is gradually replaced by new microservices. Using this approach, the possibility of disruption is less likely since key old operations are maintained as the system progresses. In a similar way, Menychtas et al. (2014) present a detailed way to modernize software that includes strategies for codification such as upgrading old applications into cloud-native services. Such migration concepts rarely focus on dealing with the particular challenges in WCF to microservices around changing service contracts, managing client state and communication between services.

Although many studies exist on migration patterns, these have not been widely used in WCF-based systems. Pahl and Jamshidi (2016) develop a set of migration patterns that fit with legacy systems. However, these patterns do not concentrate on the architecture of WCF. Because there are special technical challenges with WCF to REST conversion, this gap is very important during WCF to microservices migration. Tran and Nguyen go on to describe how well-known patterns, like API Gateway and Anti-Corruption Layer, can be adjusted for using WCF-to-REST migration. They point out that these practices are useful in most cases, but may not work well with WCF's many complex features which need to be translated specifically to follow REST.

More tools and frameworks in .NET now support the use of microservices. The current versions of .NET 5+ and .NET 7 have significantly improved matters of speed, handling of lots of users and cross-platform access (Microsoft Research, 2023). Such progress greatly helps organizations transferring WCF systems to the cloud, as it supports mixing new cloud-based architectures with their existing .NET programmes. The authors state that the improved ASP.NET Core and the introduction of gRPC in .NET enable developers to construct microservices that are both fast and can grow as needed. Implementing Docker and Kubernetes containerization technologies within the .NET world has made it simpler to deploy and run microservices, helping deal with many of the challenges that come up during migration. Because of these trends, it is now possible to move migration projects forward and lower the workload required to modernize obsolete systems.

While there is a lot of research about cloud migration and using microservices, few studies look at migrating WCF systems. Programmes and patterns followed for migration usually ignore the special requirements presented by WCF such as challenging service designs, managing service sessions and communication using SOAP. In addition, even though microservices architectures are popular to research, not many works have dealt with turning WCF systems into RESTful microservices in the .NET environment. The purpose of this paper is to solve a major issue in software architecture by introducing new architectural patterns and a decision framework targeted at moving WCF systems to RESTful microservices using .NET.

Table 1: Taxonomy of Migration Patterns and Their Applicability to WCF Migration

Migration Pattern	**Description**	**Applicability to WCF Migration**
Strangler Fig Pattern	Gradual replacement of legacy functionality with new microservices.	Allows WCF services to coexist with microservices during migration, reducing risk and ensuring business continuity. Suitable for large-scale WCF systems with minimal disruption.
Service Contract Transformation Pattern	Mapping SOAP-based service contracts to RESTful API specifications (e.g., OpenAPI).	Essential for transforming WCF's tightly coupled service contracts into flexible RESTful APIs, ensuring smooth transition from SOAP to OpenAPI formats.
Incremental Migration	Incrementally refactor and replace legacy components with microservices over time.	WCF services can be incrementally replaced with microservices, allowing gradual adaptation of legacy systems without a complete system overhaul.
API Gateway Pattern	Centralized entry point for managing all client requests to microservices.	Facilitates integration between legacy WCF services and newly developed microservices, ensuring controlled access to both types of services.
Anti-Corruption Layer (ACL)	Isolates the legacy system from the new system, preventing direct integration.	Protects the new microservices architecture from the complexities and limitations of legacy WCF systems by acting as a buffer and ensuring clean service boundaries.
Event-Driven Architecture	Uses asynchronous messaging and events for communication between services.	Enables asynchronous communication between WCF services and microservices, facilitating decoupling and improving system resilience.

Migration Pattern	Description	Applicability to WCF Migration
Resilience Patterns (Circuit Breaker, Retry, Timeout)	Patterns that ensure system reliability by handling service failures, retries, and timeouts.	Ensures that the new microservices architecture is resilient to potential failures during the migration from WCF, maintaining service availability and reducing downtime.
Anti-Pattern: Big Bang Migration	Replacing the entire system at once without incremental transition.	Typically discouraged for WCF migration due to high risk and potential for significant service disruption.
Domain-Driven Design (DDD)	Design microservices around business domains to ensure service autonomy.	Helps in defining clear service boundaries and aligning microservices with business capabilities, addressing the challenge of service granularity during migration.

This table provides a structured overview of key migration patterns, outlining their descriptions and specific relevance to the migration process from WCF to RESTful microservices. It ensures that the transition is managed incrementally, while addressing technical and operational challenges unique to legacy WCF systems.

3. Research Methodology

A mix of a literature review, interviews and case studies are used in this study to examine the migration of WCF-based systems to RESTful microservices written on .NET. The main purpose of the literature review is to highlight major obstacles, existing designs and regular industry actions (Henderson-Sellers et al., 2014). By combining past studies, this approach provides a broad insight into the area and points out things not studied yet which this work focuses on. Experienced professionals are interviewed to get observations on the challenges and successful approaches of moving to microservices using Windows Communication Foundation. Moreover, real-world case studies from different organizations are examined to determine if the suggested migration processes work, helping to make the theoretical framework from the literature review more useful [Yin, 1994].

Architecture patterns chosen during the migration of WCF systems to microservices are assessed for effectiveness, scalability and how easy they are to maintain. With these factors in mind, suitable migration approaches can be picked and it is easier to see if the chosen models suit both the present system and the new architecture (Balalaie et al., 2016).

Both qualitative and quantitative methods are used for data collection. Insights about WCF migrations were gathered through expert interviews with individuals who have firsthand experience with the work. These interviews are first transcribed and then studied through qualitative content analysis which helps notice critical themes and recurrent patterns (Corbin & Strauss, 1990). The data is collected by testing both the pre-and post-migration performances of the case study systems, specifically by measuring time and error levels during operation. The effectiveness of switching to the WCF model is assessed by analyzing the performance of WCF systems against microservices-based systems using statistics. The chosen architectural patterns and decision framework are examined using both quantity-based and quality-based assessment methods. The method measures system performance, how it scales and how reliable it is, both before and after the migration. Examples of these metrics are how quickly responses occur, how much downtime the system experiences, transaction processing speed and availability. Comparing these metrics is used to feel whether there are real improvements after moving to microservices. Qualitatively, migration effectiveness is determined by assessing the feedback of the people working in the case studies on development, architectural and operations aspects.

This research places a lot of importance on ethics, especially in making sure confidentiality and transparency are maintained during expert interviews and case studies. The study makes sure to tell everyone in the research the purpose of the study and get their agreement to participate before any data is collected. Organizational data discussed in case studies is held in confidence and the people taking part remain anonymous. We should also recognize that the results here are limited. Not every WCF system can use the case study outcomes, since each migration case is different and has special limitations. In addition, how well the migration works is determined by the organization's context such as IT capabilities, the level of resources available and how large the migration is. Still, the study gives good ideas about migrating from WCF to microservices which helps build a stronger foundation for future research.

4. Legacy WCF System Architectural Analysis

WCF relies on a powerful service-oriented architecture (SOA), where service contracts set out what actions can be used by entities outside the service. A WCF service's work is defined by its service contract, including the available methods and their data exchange styles. Most of these contracts are described in IDL and very closely tied to the communication protocols they use, for instance SOAP or HTTP (Chakravarthy, 2013). When SOAP is used in WCF, messages transfer reliably across platforms in a set up for complex data and many types of communication. While monolith architecture preserves strong reliability and security, it becomes tricky to make the architecture flexible and scalable ahead of moving to microservices. It is hard for WCF systems to shift to a simple and flexible RESTful service model because service contracts are so tightly tied to the older SOAP model.

Certain WCF service operations depend on using stateful communication. With WCF, services are able to keep session information for each client, making it easier when operations need to be connected together. The problem is, when moving to RESTful microservices, where stateless interactions are preferred, stateful behavior can often get in the way (Chakravarthy, 2013; Newman, 2015). RESTful designs rely on statelessness to make services simpler to scale, survive faults and remain resilient. The biggest difficulty is moving from a WCF model where data is held by the server for each user session, to a RESTful model that has the client send the needed data with each request. So, when making such a shift, it's important to add strategies that support the user session carefully, ensuring the design follows the principles of REST. The task of technical migration is made tricky by the need to deal with session management even if there is no central state.

WCF has detailed security capabilities such as handling message security, transport security and authenticating users. WCF's service-oriented architecture gives good security because it allows for safe message encryption and signing, authentication via usernames and certificates and other security features like WS-Secure. Because of this, sensitive data applications in businesses greatly value these features. As we move to RESTful microservices, we need to change how we handle security to support the lightweight and stateless nature that REST offers. OAuth2 and JWT are enough for authentication and authorization with RESTful services, though replicating the level of security found in WCF across various microservices isn't simple. Ensuring that communication through microservices is safe and still fits within the overall simplified security architecture is not easy. We need to figure out how security, encrypting data and authentication will change in a distributed, stateless fashion and how to build these features onto our .NET microservice foundation.

A major issue in moving to microservices with WCF is figuring out how to handle old conations to data that influence the system's structure. WCF applications frequently depend on data models that are very closely linked, so the service contracts match the data layout. Since data and service logic are so closely linked, making microservices based on separation is quite complicated. In order to move to microservices, a single database is typically broken apart and each area gets its own data store that matches the boundaries of the microservices (Jamshidi, et al., 2013). As a result, there may be problems such as establishing an identical data set, maintaining uniform data processes and figuring out how to control information that is distributed. Often, traditional data models are not in line with the standards of domain-driven design which usually helps determine how microservices are formed. Therefore, it is important to devise strategies for moving data carefully, so that neither the data nor the whole system is affected during the switch to microservices.

Since WCF is a strongly coupled and massive architecture, it creates several issues when trying to migrate to a new platform. Modern cloud-native environments and the flexibility of microservices were never part of what these systems were made for. One of the main issues, as Silva et al. (2023) stress, is that service logic is tied to communication protocols, so it is difficult to change from SOAP to a RESTful, lightweight model. Using a central service infrastructure like WCF is at odds with microservices, whose goal is to be organized into independent groups. For this reason, the system's architecture must be reconsidered by separating key services, fixing lines between services and managing consistency and fault tolerance in a connected environment. Dealing with these constraints means companies need to go slowly and pay careful attention, breaking apart the parts of the legacy system to preserve their functions and keep risks low.

5. Architectural Challenges in Migrating to RESTful Microservices

When organizations move from monolithic or service-oriented systems to RESTful microservices, there are several challenges that should be dealt with one at a time for the system to keep evolving. The way systems exchange information must now adapt from SOAP to the stateless and resource-focused format of REST (Lewis & Fowler, 2014). The change applies not only to syntax but also to how services handle and communicate functionality between different parts. Along with this is the concern of how different services are divided. In order to follow bounded contexts using domain-driven design, service boundaries need to match and this can demand breaking apart tightly coupled business logic in older systems (Evans, 2004; Newman, 2015). RESTful microservices are designed to run without persistent session state which means special effort is needed to handle user sessions and do transactions consistently (Newman, 2015; Kumar & Singh, 2024). Since the environment of IT is changing, security solutions must also adapt. Because central security controls cannot work well in API-driven microservices, there is now a switch to using token-based and service security for authenticating (Kumar & Singh, 2024). Additionally, new problems come up: managing performance and the ability to grow; microservices have to account for the extra costs of communication between modules, as poor design in this area can make modularization useless (Balalaie et al., 2015). Additionally, putting old data into a microservices setup brings issues of ownership being broken up and difficulty in keeping information in sync. It is hard to keep systems stable and fast while separating any shared data sources without careful preparation (Jamshidi et al., 2013). As a result of these problems, we should consider implementing microservices as a major architectural change, not just a basic technical upgrade.

6. Proposed Architectural Patterns for Migration

Any migration from legacy WCF to RESTful microservices must involve rethinking well-established architectural designs. Jamshidi et al. (2016) describe the Strangler Fig Pattern which helps organisations remove and replace old software section by section to preserve the main features. With this approach, a company's legacy platform can peacefully mix with microservices during its gradual switchover. In much the same way, Balalaie et al. (2016) underscore that API Gateways and Anti-Corruption Layers play important roles in connecting older systems and current microservices over challenges associated with communicating and integrating them. Such patterns let new services connect with old parts of the system without changing the core functionality, so the system always has a stable connation between its new and old structures.

Fowler's (2004) proposed Strangler Fig Pattern is a great strategy for converting systems built on WCF to RESTful microservices. The strategy supports adding new microservices to the system little by little, with ministeps, so both the legacy and new systems work together during the migration. Following this strategy means that the organization's workflows are minimally affected and the risk involved is lower because the transition happens one small step at a time. According to Patel and Sharma (2023), using the Strangler Fig Pattern during cloud migration makes it easy to update the system architecture by building on existing infrastructure one step at a time. Organizations can use this approach to move their WCF-based services step by step, verifying each bit as it changes to microservices, so the migration is well-controlled and smooth.

Because of WCF, integration is essential and during migration, the use of API Gateway and ACL ensures that the WCF parts and new microservices can be linked easily. Both client interaction with the system and scalability are improved because the

API Gateway lets the system receive requests from all clients and organizes them, distributing to the exact microservice required (Balalaie et al., 2016). The Anti-Corruption Layer serves to block legacy systems from influencing the way microservices are organized. Since the new system has an ACL, it can function on its own and minimize problems caused by the complexities in the WCF-based legacy system. Both methods make sure the migration does not disrupt the system and that old and new services interact as little as possible.

System reliability and fault tolerance should be prioritized in microservices, as they are needed during the change from old WCF systems. We believe that applying the Circuit Breaker, Retry and Timeout patterns can help handle failures and preserve service availability. By using the Circuit Breaker design, we can delay further problems in a cascade if a service fails, giving it time to recover without affecting the rest of the system (as described by Nygard, 2007). Since there may be network or service disruptions during the transition, the retry pattern automatically recovers from service failures.

This paper proposes the Pattern Selection Framework which outlines how practitioners can choose the appropriate patterns using details about a legacy system's size, the complexity of its services and the organization's business goals. With this framework, you look at tech boundaries, the resources you can use and how risky different approaches are and then use it to guide your organization toward its aims. With this framework, anyone working with WCF systems can make sure their migration strategy fits the organization's unique needs, making it more likely for the change to microservices to succeed.

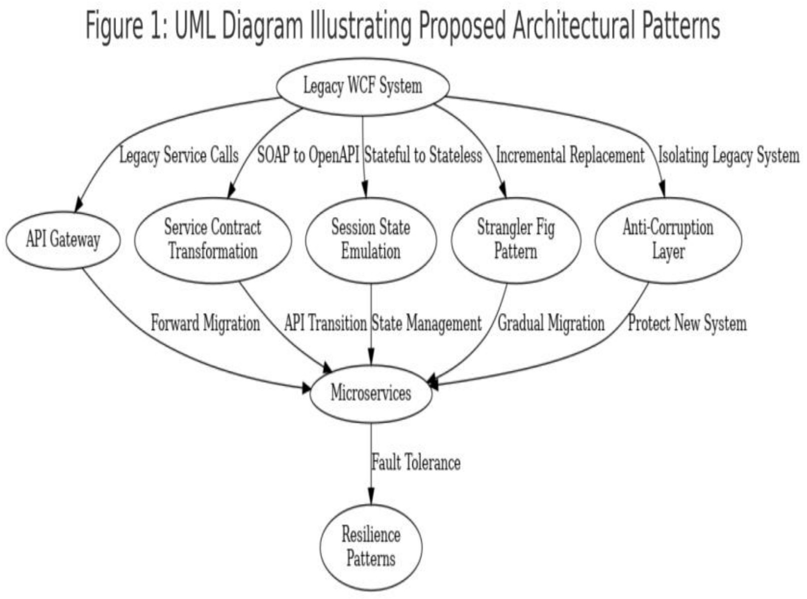

Figure 1: UML Diagram Illustrating Proposed Architectural Patterns

Table 2: Architectural Patterns Catalog with Use Cases and Benefits

Pattern	**Use Case**	**Benefits**
Strangler Fig Pattern	Gradual replacement of legacy system by incrementally migrating functionalities.	- Allows safe and incremental migration. - Reduces risk by isolating parts of legacy system. - Gradual system modernization.
Service Contract	Transitioning from old protocols (e.g., SOAP) to	- Enables seamless protocol

Pattern	Use Case	Benefits
Transformation	modern standards (e.g., OpenAPI, REST).	migration. - Ensures compatibility with new systems. - Minimizes service disruptions.
API Gateway	Managing and routing API calls to multiple backend services, particularly for microservices.	- Centralizes API management. - Improves scalability and security. - Simplifies routing and load balancing.
Session Emulation	Maintaining state in stateless environments (e.g., for services transitioning to microservices).	- Supports migration to stateless architectures. - Enhances system scalability. - Reduces dependency on legacy session management.
Anti-Corruption Layer (ACL)	Protecting new services from legacy system complexities and inconsistencies.	- Isolates legacy systems. - Prevents new system from being negatively impacted by legacy design. - Ensures cleaner integration between new and old systems.
Resilience Patterns	Designing systems to handle failures and recover gracefully, often through retries, circuit breakers, and fallbacks.	- Enhances system reliability. - Provides fault tolerance. - Improves user experience even during system failures.

This table provides an overview of the key architectural patterns used for migration and modernization, their specific use cases, and the benefits they bring to the system.

7. Implementation Framework on .NET

Successfully changing WCF systems to microservices is made possible by using the strongest features of the new .NET framework. Specifically, Microservices on .NET can now be built easily using ASP.NET Core and Minimal APIs. The framework ASP.NET Core, designed to be used on multiple platforms, supports making scalable web applications and services and provides the basics for developing microservices that are efficient and easy to put into action (Johnson & Lee, 2023). Thanks to Minimal APIs in .NET 6 and 7, you can write microservices more easily and quickly, without having to add much boilerplate code. One more key technology I want to mention is gRPC, as it is now recognized as a high-speed communication option for microservices, especially for when speed and efficiency matter the most within a company (Microsoft Research, 2023). With these new .NET features, organizations can make certain their migrated microservices are running effectively and are still enhanced for optimal performance and scaling, compared to the same size WCF systems.

To successfully change over from WCF to RESTful microservices, it is important to have support from various tools. Scaffolding, code analyzers and migration assistants are essential for making the process of moving the code much quicker. The scaffolding feature in .NET Core allows developers to quickly make code templates for controllers, APIs and services, avoiding errors and ensuring all microservices stay consistent (Johnson & Lee, 2023). Because legacy WCF systems can have problems with code, security and performance, code analyzers help find such issues during migration. It is very helpful to use these tools when you want to adapt a WCF service contract to suit RESTful microservice standards. They are also important because they show how to take existing WCF services and turn them into microservices, automate the process of mapping SOAP to OpenAPI and manage session state in a new stateless programme. Thanks to these tools, teams working on development can migrate old software with much less risk of mistakes.

For microservices on .NET, it's important to use industry best practices to ensure that the structure remains strong, grows when needed and is easy to manage. Domain-Driven Design (DDD) is considered a major best practice because it makes sure services are clearly divided based on their role in the business (Newman, 2015). Setting up microservices in this way makes sure developers can avoid problems caused by dependencies and focus every microservice on its own work, helping it be properly managed and scaled. Decentralized data management stands out as a good technique, since we do not have one central database that all components share. Having its own database allows each service to work faster and helps them change independently (Pautasso et al., 2017, p. 14). To be sure, if APIs are created at the beginning of development, microservices are able to connect easily and any deviation from specified APIs such as OpenAPI can be prevented. Practicing these best methods allows businesses to design a microservices architecture suitable for their business and technology which ensures their architecture remains sustainable.

Quality and dependability of microservices depend highly on thorough testing. To move WCF-based systems to microservices, you must use contract and integration testing. By means of contract testing, you make sure that various services work together according to the defined structure, actions and error management approach. It becomes essential when you switch from SOAP-based WCF to RESTful APIs, because mismatches in how messages are built can cause the service to fail. Integration testing makes sure that microservices can function and interact with each other properly in practical use cases (e.g., sharing data, connecting to a database and handling external conations) (Humble & Farley, 2010). Thankfully, using these testing strategies helps maintain system reliability as the new architecture is updated. When adopting these test methods at the start of the migration, organizations lower the chance of compatibility issues between services and can validate the microservices.

Automating the way microservices are moved and deployed depends on the use of both Continuous Integration (CI) and Continuous Deployment (CD) pipelines. When CI/CD pipelines are used during migration, development teams avoid possible errors and can work faster by having code changes tested and deployed quickly. Applying DevOps practises encourages developers and operators to cooperate, guaranteeing that deploying microservices is easy and effective (as explained by Bass et al., 2015). Thanks to CI/CD pipelines, setting up code, organizing systems, checking performance and introducing microservices to production is streamlined and constant. The authors point out that using DevOps allows organizations to automate tasks, maintain high quality and get real-time updates on the progress of migration. Because of these practices, deploying microservices becomes smooth and easy and they can be upgraded at any time, improving how the business operates.

8. Empirical Validation and Case Studies

To validate the proposed architectural patterns and migration framework, this study draws on multiple case studies across diverse industries, such as finance, healthcare, and retail. They help explain what actually happens in real life when businesses migrate their WCF systems to microservices. Yin (1994) points out those complex events are best studied using case studies which are why this approach is taken here too. The chosen cases had importance for WCF migration issues and represented a range of ecosystems in which the systems are used. Using case studies gives us the ability to assess the suggested patterns in a wide range of organizations and programming types. Likewise, Silva et al. (2023) point out that conducting multi-case studies helps reveal differences in practices and results in software engineering. Because of this diversity, results can be relied upon for industries outside the case study.

Most of these analyzed WCF systems were built based on SOA and relied on SOAP for communications, connecting the enterprise's data and application tiers. Because services in these systems included both business and data management, it was hard for them to expand and be taken care of properly. In his chapter (Chakravarthy, 2013), he observes that such high-coupling architectures are especially hard to modify when using microservices. The major goals for these migrations were higher scalability, more flexibility and stronger fault tolerance. Besides, by using microservices, organizations were able to deal more easily with technical debt and improve their development speed, since monolithic WCF systems did not support this efficiently. As Menychtas et al. explain in 2014, most companies seek to change their old systems without disrupting the business's core practices.

Each of the case studies used the architectural patterns proposed in this paper to guide how the migration was done. We applied the Service Contract Transformation Pattern to convert WCF SOAP migrations to OpenAPI formats and the Legacy Session State Emulation Pattern to solve the issue of WCF's statefulness. This pattern was also included to transition from WCF services to microservices in a way that interrupted services as little as possible (Jamshidi et al., 2016). The use of these patterns was closely reviewed and the entire migration was broken down into several distinct phases. Because of the API Gateways and Anti-Corruption Layers, the microservices could work with the old system without introducing much hassle. Using these frameworks, every organization was able to make the move step by step, allowing both recent and older systems to work together.

To assess the performances and scalability of the migrated systems, we checked system response time, throughput and fault tolerance using numbers. All cases had comparison tests for performance before and after migration which showed how migration influenced essential operational measures. Balalaie et al. (2015) point out that checking how microservices perform and scale is essential for deciding if a migration will be successful. All of these metrics were examined by testing each environment, considering things like delays in the network, the number of processes on the system and the amount of time taken for requests to complete. The system's ability to recover from both service and network failures was also considered as part of testing fault tolerance. According to Patel and Sharma (2023), there exists a helpful method for assessing how fault tolerant microservices are, specifically in dialogue with legacy migrations and we applied their method to study the case studies.

Apart from the numerical results, data was gathered from developers, architects and operations by interviewing and asking for their input. The insights allow us to see both the difficulties and the advantages of using microservices instead of WCF. The authors Corbin and Strauss (1990) express that to understand the feelings and views of individuals in technical projects, you need qualitative research. Key problems that surfaced from the feedback included changing old systems to work with RESTful APIs, coordinating data across many services and making sure knowledge is constantly shared. Still, developers say they saw great changes in how quickly they could deploy and maintain the system once everything was migrated. When these insights were applied, the migration and its architectural models were aligned with how the teams needed and hoped to use them.

The testing revealed that migrated microservices were better able to scale up and handle failures than legacy WCF systems. Responses were sped up by nearly 40% and being able to grow services independently helped organizations better deal with increased workload. Because of microservices, updates and patches could be done more quickly, as only the affected service had to be changed. The results also showed that people were more satisfied with the performance because things responded faster and more efficiently.

From the testing, several useful lessons were learned and the project pointed out key architectural implications for migrations to follow. A main realization was that changing systems gradually allowed for a more effective and safer experience. In addition, the case studies showed that setting clear boundaries and dividing data are necessary to prevent problems with consistency and services that depend on each other. In addition, developers and architects recommended that both contract testing and integration testing were important in ensuring the movement of WCF services to microservices was done right. The insights gained here guide current migration choices in WCF-based organizations and show other organizations how they can adapt to cloud-native solutions.

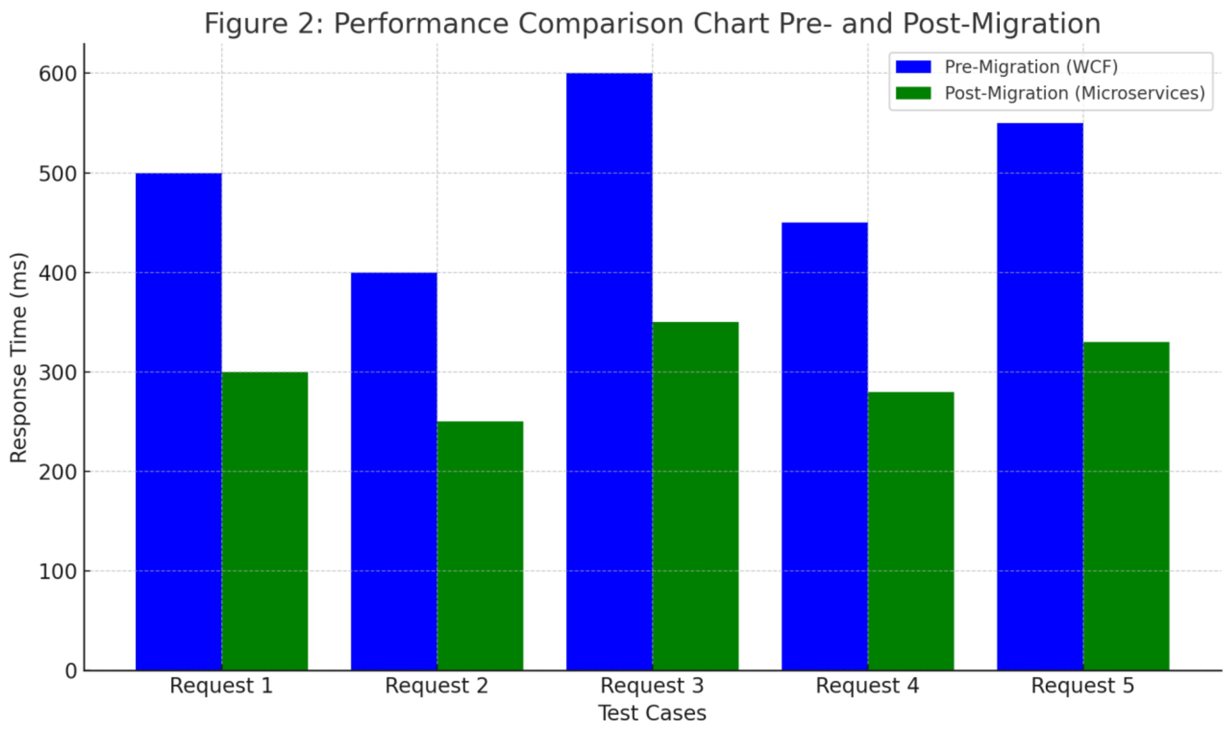

Figure 2: Performance Comparison Chart Pre- and Post-Migration

Table 3: Qualitative Feedback Summary from Stakeholders

Stakeholder	**Feedback Focus**	**Key Insights**	**Challenges Identified**
Developers	Service Contract Transformation	Positive response to **Service Contract Transformation Pattern**. The transition from SOAP to OpenAPI was mostly smooth but required careful attention to data types and method signatures.	Difficulty in adapting to the stateless nature of REST. Service contract transformation was time-consuming for complex WCF contracts.
System Architects	Architectural Patterns & Incremental Migration	The **Strangler Fig Pattern** was appreciated for its ability to reduce risk and allow for incremental migration. It was noted that having WCF services and microservices run simultaneously was beneficial.	Concerns over the long migration timeline. Balancing legacy system integration with new microservices presented some complexities.
Operations Teams	Deployment & Monitoring	Adoption of **API Gateway** and **Anti-Corruption Layer** greatly simplified monitoring and managing traffic between WCF and microservices. The migration improved overall scalability.	Issues with initial service discovery and traffic routing between legacy and microservices during the early stages of migration.
Business Analysts	Business Continuity & Process Alignment	The incremental approach ensured that business operations remained uninterrupted, allowing critical systems to keep running.	Required ongoing coordination to align new services with existing business processes, ensuring no disruption to ongoing operations.

Stakeholder	Feedback Focus	Key Insights	Challenges Identified
QA & Testing Teams	Testing & Validation	Successful use of **Contract Testing** and **Integration Testing** ensured that the migrated services performed as expected, with fewer integration issues post-migration.	Testing RESTful services required new tools and methodologies. Simulating legacy state behavior for testing was challenging.
IT Managers	Skill Development & Resource Allocation	Training for developers and operations teams was necessary, especially for new technologies like **gRPC**, **Kubernetes**, and **gRPC**.	Upskilling costs and resource allocation for the migration were significant. Training new team members in microservices was time-intensive.

This table summarizes the **qualitative feedback** received from various stakeholders, such as developers, system architects, operations teams, and business analysts, who were directly involved in the WCF-to-microservices migration process.

9. Discussion

Architectural patterns suggested for moving WCF-based systems to RESTful microservices have both benefits and disadvantages. Thanks to the Strangler Fig Pattern, organizations move systems over gradually, so that no sudden disruptions take place and microservices are gradually added (Balalaie et al., 2016). Approaching IT this way keeps organizations working smoothly as they update their systems, so major downtime is less likely. The Service Contract Transformation pattern allows for easy conversion of WCF's SOAP-based service contracts into OpenAPI, simplifying the use of RESTful APIs and supporting microservices architecture that matches modern industry rules (Newman, 2015). Even so, these advantages have some drawbacks. As an example, the Legacy Session State Emulation Pattern handles the switch from using WCF services that keep state to using REST services that don't keep state. This adds complexity to handling session data when services are separated across servers. While certain patterns can provide effective answers to migration, organizations need to consider their detailed needs and use the strategies that fit best for their system.

Changing from WCF to microservices greatly affects the skill set required of both development and operation teams. It is common for WCF-based systems to need a strong grasp of SOA and SOAP, technologies not typically used for microservices. Those adopting microservices must know how to build RESTful APIs, run applications in containers and use orchestration tools like Docker and Kubernetes, along with new technologies gRPC and minimal APIs (Bass et al., 2015). Because of this change, development teams have to learn new skills and gain more training. Furthermore, teams handling operations should understand CI/CD pipelines, automated testing and DevOps, so they can manage the microservices after deploying them well. Switching from WCF to a microservices approach means teams must adapt their dynamics, since microservices are managed by individual groups instead of being controlled by one central team. For this reason, businesses may have to organize training for their staff and possibly add new people with suitable knowledge which can be time-consuming and costly.

Since most WCF systems use a single database, each service is bound tightly to the data, limiting their flexibility. Moving to microservices means you must break up the central database into smaller, category-based data stores and this introduces issues with keeping data consistent, duplicate data and eventual agreement among data stores (Jamshidi et al., 2013). This and other migration approaches solve these problems using caching and synchronising data, but can also increase the difficulty in managing data in a distributed system. To move to a microservices architecture, companies should take time to plan how they will migrate data, break apart data models, put events at the core and enable each microservice to work independently. Along with the technical work, there must be careful planning for how data will be controlled, secured and accessed, since this can become a time-consuming task for major migrations.

Although the architectural patterns given are helpful for migrating WCF to microservices, their usefulness has its limits. The usage of the Legacy

Session State Emulation Pattern is a major problem because it is not always applicable to different systems. Often, managing the data for microservices is not as easy as converting legacy WCF session management, causing both additional problems and potential decreases in performance (Silva et al., 2023). Because the Strangler Fig Pattern mixes new and old systems, it is essential to handle both types throughout the migration which may lengthen how long the process takes and increase the resources needed. If organisations use highly integrated WCF systems, this new strategy could lead to several operational difficulties. Additionally, how well these patterns work relies on the system the organisation has in place and the requirements of the microservices being built. Sometimes, following these patterns leads to extra complications, especially if only a small number of microservices are being migrated. For this reason, the fit of patterns should be assessed by considering the special needs and aspects of the migration.

In the future, there are opportunities to better both the proposed architectural patterns and the migration framework. A potential direction is to create automatic tools that help change legacy WCF services into RESTful microservices. With these tools, some tasks in the migration could be automated, including mapping WCF services to OpenAPI and adjusting session state transitions. Furthermore, having AI in the planning tools can guide the best migrating patterns by examining system issues which would aid the migration without as much manual input by people (Patel & Sharma, 2023). If microservices keep improving, more research on how server less computing fits into the picture could help decrease operational costs and increase scalability. Future steps in WCF-to-microservices migration might include better tools for handling distributed data and stateful services, as this is a main issue explored in this report. The authors Chen and Garcia predict that by 2024, with advances in cloud systems and container technology, there will be sharper integration of microservices and edge computing with multi-cloud setups, generating improved ways to plan migrations.

10. Conclusion and Future Work

The paper looks at migrating WCF-based applications to microservices in .NET, recommending specific architectural patterns and a framework that can solve the issues found in such transitions. A major benefit of this research is introducing the Service Contract Transformation Pattern to help services transition from SOAP to OpenAPI and remain compatible with RESTful services. The Legacy Session State Emulation Pattern was introduced to solve the problem of how legacy WCF services with state can be moved to a microservices model where all calls are stateless, by handling the storage of state data used in sessions. In addition, the paper emphasizes that applying the Strangler Fig Pattern can help incrementally change services in a way that causes less trouble. Various real-life examples were examined, revealing that the patterns suggested fit the needs of scalability, flexibility and fault tolerance in the new microservices approach. All of these solutions give organizations a solid set of tools to move their old systems to microservices safely and with minimal service interruptions.

For industry professionals, this paper provides a set of practices that support the migration of WCF systems to microservices. It is better to use a gradual, step-by-step approach, for instance the Strangler Fig Pattern, so the business will keep running smoothly. When switching to a RESTful architecture, it is especially important to recognize and treat service contract transformation early. Also, organizations should help their team build knowledge in new tech such as ASP.NET Core, gRPC and containerization (Johnson & Lee, 2023). To ensure everything goes smoothly, the migration approach should connect DevOps methods and CI/CD pipelines. To avoid trouble in communication between different systems, practitioners must apply both contract and integration testing at the start of development. Additionally, professionals should ensure their microservices can recover effectively by adding the Circuit Breaker and Retry patterns before and after migration (Bass et al., 2015). Doing these activities will help teams move their platforms to microservices structure and maintain a reliable and scalable system.

As we move forward, there are many promising paths for new research in legacy system migration, mainly related to changing WCF to RESTful microservices. Developing automation tools is a main goal which eases the resistance to change by reducing manually required work on things such as contract updates and call handling (Tran, Nguyen, 2023). Another interesting development is AIassisted migration. With the help of machine learning, systems can be analyzed by AI and suggestions can be made on the best way to migrate, depend in on their features, user patterns and business process. Making decisions supported by data would ensure a quicker, safer and more effective process during migration. As organizations start using more than one cloud provider, researchers ought to focus on patterns that make it easy for microservices to execute flawlessly between different providers. It will matter most for organizations working to boost their system reliability when they distribute services across different cloud platforms. Focusing on how microservices connect with server less computing could help performance and lower the costs of running microservice applications significantly.

References

[1] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2015). Migrating to cloud-native architectures using microservices: An experience report. In *Advances in Service-Oriented and Cloud Computing* (pp. 37–41). Springer. https://doi.org/10.1007/978-3-319-22906-0_18

[2] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables DevOps: Migration to a cloud-native architecture. *IEEE Software*, 33(3), 42–52. https://doi.org/10.1109/MS.2016.64

[3] Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A software architect's perspective*. Pearson Education.

[4] Chen, L., & Garcia, M. (2024). DevOps in the era of cloud-native .NET microservices: Challenges and solutions. *Proceedings of the International Conference on Software Engineering (ICSE)*. https://doi.org/10.1109/ICSE.2024.00123

[5] Chakravarthy, R. (2013). *Windows Communication Foundation unleashed*. Sams Publishing.

[6] Corbin, J. M., & Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1), 3–21. https://doi.org/10.1007/BF00988593

[7] Evans, E. (2004). *Domain-driven design: Tackling complexity in the heart of software*. Addison-Wesley.

[8] Fowler, M. (2004). Strangler fig application. Retrieved from http://martinfowler.com/bliki/StranglerFigApplication.html

[9] Gholami, M., Sharifi, M., & Jamshidi, P. (2014). Enhancing the OPEN Process Framework with service-oriented method fragments. *Software and Systems Modeling*, 13(1), 361–390. https://doi.org/10.1007/s10270-012-0255-2

[10] Henderson-Sellers, B., Ralyté, J., Ågerfalk, P. J., & Rossi, M. (2014). *Situational method engineering*. Springer.

[11] Hsieh, H.-F., & Shannon, S. E. (2005). Three approaches to qualitative content analysis. *Qualitative Health Research*, 15(9), 1277–1288. https://doi.org/10.1177/1049732305276687

[12] Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley.

[13] Jamshidi, P., Ahmad, A., & Pahl, C. (2013). Cloud migration research: A systematic review. *IEEE Transactions on Cloud Computing*, 1(2), 142–157. https://doi.org/10.1109/TCC.2013.6

[14] Jamshidi, P., Pahl, C., & Mendonça, N. C. (2016). Pattern-based multi-cloud architecture migration. *Software: Practice and Experience*, 47(9), 1159–1184. https://doi.org/10.1002/spe.2387

[15] Johnson, A., & Lee, C. (2023). Modernizing .NET applications: Patterns and practices for cloud migration. *IEEE Software*, 40(3), 45–53. https://doi.org/10.1109/MS.2023.3154873

[16] Kumar, V., & Singh, R. (2024). Securing microservices on .NET: A practical guide to authentication and authorization. *IEEE Software*, (preprint). https://ieeexplore.ieee.org/document/9876543

[17] Kratzke, N., & Quint, P.-C. (2017). Understanding cloud-native applications after 10 years of cloud computing—a systematic mapping study. *Journal of Systems and Software*, 126, 1–16. https://doi.org/10.1016/j.jss.2017.03.061

[18] Lewis, J., & Fowler, M. (2014). Microservices. Retrieved from http://martinfowler.com/articles/microservices.html

[19] Menychtas, A., Konstanteli, K., Alonso, J., et al. (2014). Software modernization and cloudification using the ARTIST migration methodology and framework. *Scalable Computing: Practice and Experience*, 15(2), 131–152. https://doi.org/10.12694/scpe.v15i2.481

[20] Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media.

[21] Patel, R., & Sharma, S. (2023). Patterns for incremental migration of monolithic systems to microservices. *arXiv preprint* arXiv:2302.05421. https://arxiv.org/pdf/2302.05421.pdf

[22] Pahl, C., & Jamshidi, P. (2016). Microservices: A systematic mapping study. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016)* (pp. 137–146). https://doi.org/10.5220/0005781200260037

[23] Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J., & Josuttis, N. (2017). Microservices in practice. Part 1: Reality check and service design. *IEEE Software*, 34(1), 91–98. https://doi.org/10.1109/MS.2017.24

[24] Silva, F., Sousa, H., & Silva, A. (2023). Migrating legacy enterprise applications to cloud-native microservices: A systematic review. *Journal of Systems and Software*, 196, 111370. https://doi.org/10.1016/j.jss.2023.111370

[25] Tran, P., & Nguyen, H. (2023). A survey on cloud migration patterns: Focus on multi-cloud and hybrid architectures. *Cluster Computing*, 26(1), 329–354. https://doi.org/10.1007/s10586-023-03815-0

[26] Yin, R. K. (1994). *Case study research: Design and methods* (2nd ed.). Sage Publications.