A Domain-Driven Design Approach for Micro-Services using Web Services

Dhruv Seth[1]

Solution Architect, Walmart Global Tech, California, USA
Email: *er.dhruv08[at]gmail.com*

Abstract: ***Microservices have faced developments in terms of design and enhancement of their service provision scope. The capacity to address and enable an appropriate modelling of the microservices creates a demand to use domain driven design (DDD) relying on web services to enhance the appeal. DDD provides several values to the microservices, enhancing the possibility of a distinctive and an even greater outcome when using software. Key steps that help to integrate DDD to the microservices includes strategies such as mapping bounded contexts to microservices, designing domain models for microservices, ensuring communication and data integrity and implementing communication between domain services. These have to be conducted with keen address to security and data privacy regulations. The collaboration and communication between microservices does not have to be hampered but adjusted to enable sustainable and appropriate scope of ensuring beneficial engagement at all levels. Furthermore, implementing the DDD approach creates a distinctive appeal that helps in structuring and enabling better performance scope for every channel of registering and addressing performance as it comes along within the desired functionalities.***

Keywords: DDD, Microservices architecture, web services

1. Introduction

a) *Definition of Domain Driven Design*

Domain Driven Design (DDD) refers to the software development approach which concentrates on modelling and handling problem domain as core to the development process. The domain drive design ensures that it can handle different approaches to ensure an increasingly beneficial scope and level of addressing critical instructions to achieve and enable supportive advancement to every category of design [1]. Fig.1. indicates different factors that define the DDD, such as:

1) Bounded Context: DDD works by ensuring that the large and complex domains can be divided into more manageable and smaller contexts. Every bounded context provides an area of domain that has individual models and rules. They help to avoid inconsistencies and conflicts between various parts of the system.
2) Ubiquitous Language: DDD applies a common language shared between developers and domain experts. This approach enables the combined approach and understanding of every party, ensuring that they deal with any possibilities for misunderstandings [2].
3) Continuous Refinement: DDD is built on the framework that domain has to evolve and encompass various changes over time. The improvement and advancement have to be conducted by experts to cater for changes within the business and address core adjustments to any level of achieving better appeals.
4) Domain Model: The domain model is the core of the design, enabling a key capacity to address business logic and regulations relating to the domain. The approach works with object-oriented programming which allows for an understanding of the domain functionalities and possibility to create a much better engagement to achieve a remarkable outcome [3].

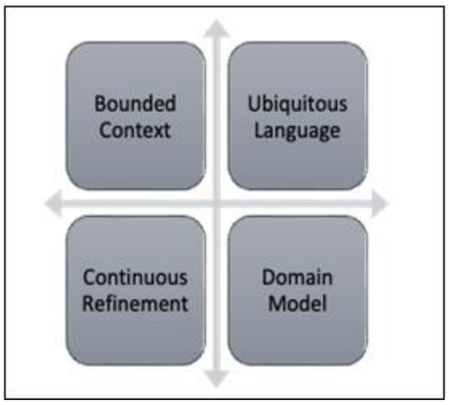

Figure 1: Components of Domain Driven Design

b) *Explanation of Microservices Architecture*

Microservices architecture is a process in software development where an application has loosely coupled services that can be deployed independently. Additionally, the approach is enabled with having smaller and more specialized services which conduct specific functions and they come together to assist in achieving the desired objective. Further Fig.2. highlights features of the microservices architecture includes:

1) Service Isolation: Services are isolated from one another, ensuring that they can operate independently and a single service does not influence the other.
2) Scalability: Services can be scaled based on the demand to assist in achieving the desired goals and outcomes at all times [4].
3) Diversity: Different technological appeals can be used within each service provision, enabling a greater attention to marking progressive flexibility of the system to achieve their demands. The diversity and flexibility contend to have a greater address and channel of addressing requirements of any service.
4) Decentralization: This indicates that every service can be independently deployed and can work without interference with the other. This approach caters for the categorical adjustment in remarking and ensuring critical

modelling of every distinctive value to achieve the right appeals and autonomous service provision as demanded [5].

5) Inter-service communication: Microservices have to communicate and engage one another in addressing the overall functionality. Communication can be conducted through HTTP/REST and it helps to ensure that there are interactions towards achieving a designed outcome. Message Queues also assist to convey communication as desired.
6) Domain Driven Design (DDD) Alignment: Services have to be aligned with DDD principles and this enables each service to have a specific context within the domain. The approach creates a better understanding and promotion of a domain-centric development approach [6].

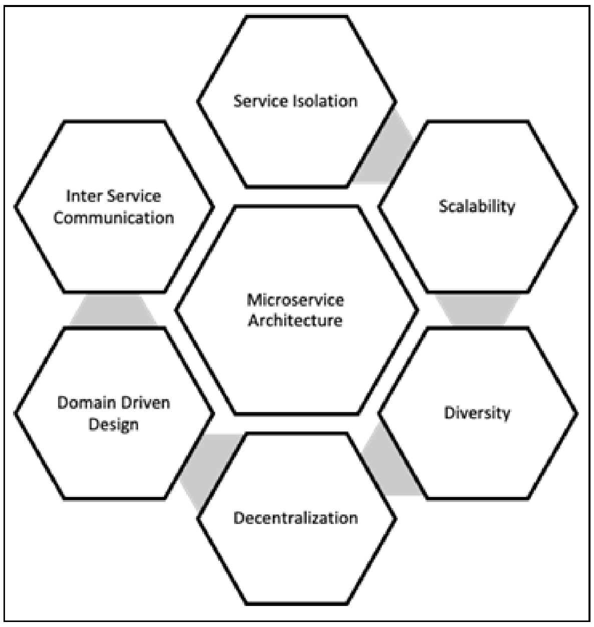

Figure 2: Features of Microservice Architecture

c) *Importance of Web Services in Microservices architecture*

Microservices architecture has to work with web services to ensure flawless functionality. Web Services have the main benefit of ensuring critical communication and interaction between every service as provided. Their core benefit stems from the capacity to align to several values within the system. Such values include:

1) Service discovery and registry: Web Services have features that assist in locating and connecting to other services with much ease and flexibility [7].
2) Loose Coupling: Web Services ensure that the microservices can maintain their independence and avoid reliance on internal engagement to achieve their functionality. This enhances flexibility and maintainability of every provided value.
3) Inter Service Communication: Web Services offer a way to ensure that services can communicate and have the best point to facilitate data exchange and functionalities that remark progressive management of their functions.
4) Security and Authentication: Web services ensure security which can enhance the applicability of the microservices. Engagement in aspects such as authorization and encryption enable achievement of the best outcomes in whatever means necessary.
5) Scalability and Load Balancing: Web Services create the chance to scale operations on individual microservices [8]. It also creates the chance to conduct load balancing on individual platforms and creating the most meaningful instruction in adjusting and enabling sustainable service offering to the domains and microservice segments.

d) *Challenges in Microservice Development*

Microservice architecture has several challenges which affect the possibility of flawless design and engagement. The challenges include:

1) Service Orchestration and Choreography: Handling service coordination between micro services is a tedious and demanding process. The use of central controller assists to ensure coordination and decentralized communication in a way that governs the provision and handling of individual approaches to deal with the execution of key duties. The communication has to be conducted in a way that ensures every pattern can collaborate without the central orchestrator. These engagements have complexities and have to be selected with regards to having the best appeal in order to ensure an instrumental point of engaging and achieving the desired value.
2) Testing Complexity: Microservices have to be tested to ensure they can be understood and every functional angle addressed to achieve the most remarkable outcome. The testing requires setting up testing environments and managing them to ensure an appeal to the right direction [9]. The entire process of setting end to end tests is time consuming and demands an engagement that will contend to having series of adjustments to enable sustainable engagement.
3) Distributed Data Management: Microservices have their own databases that store information from their operations. The challenge comes in managing the distributed data and ensuring consistency and synchronization of the services to achieve a remarkable outcome in every essence. Working within the framework of appealing to the complex data handling and management leads to greater appeals in providing multiple services and having an update of the shared data to achieve the best definition and identity in all aspects.
4) Service Discovery and Communication: The dynamic environment of microservices is challenging and demands a robust approach to help in easing service communication. The microservices have to enable seamless communication, achieving a reliable and appropriate step in addressing different needs within the system. The independent need for communication and approaches to work within every step of the way crafts the best channel to deal with issues such as latency, delivery guarantees and network failures. These are therefore the best platforms and points of ensuring an instrumental capacity to have the best communication appeals whenever demanded [10]. Thus, the resilient communication patterns such as retries and circuit breakers have to be used to foster fault tolerance and appeal to robustness of the entire system.

5) Security: Microservices have to be secured to ensure that they have the best platform and point of addressing their demands at all points. The security of the microservices therefore work towards ensuring data protection, authentication, communication encryption and authorization of every microservice. The approach dwells on the possibility to create and measure up to the modelling of threat-resistance in core ways that enhance an appeal to better functionality. Resilience to injection attacks and breaches hold a great relevance for the security management and modeling instance, where every capacity to operate can be provided to achieve the most sustainable appeal for all.
6) Use of Continuous Integration/Continuous Deployment: Automated deployment pipelines have the capacity to ensure suitable changes for every service that they offer.CI/CD has the capacity of handling dependencies and providing the best versioning as well as scaling of the activities. These approaches ensure suitable development to the level of administering reliable outcomes in whatever demands are needed.
7) Monitoring and Observation: Distributed systems within microservices demand a complex approach to monitor and register a critical way to handle their interactions. The modelling of the large number of services and addressing their engagement ensures that there are core advances in marking progressive needs in collecting and looking into logs and traces to achieve the best level of monitoring [11]. The right monitoring appeals therefore help in detecting anomalies and creating anomalies, which can be addressed to help in optimizing the performance and creating a reliable and suitable system.

Challenges in handling microservices stem from their architectural design. The decentralized functionality begs for more advances in remarking engagement and addressing core developments to assist in targeting and managing their adjustments whenever needed. The microservices have to be scaled and ensure agility in functionality, so they can help to leverage against these pertinent challenges. Achieving the full potential of the design demands a critical engagement of tools and technologies that will bring about an instrumental categorization and engagement to have the best outcome in all aspects.

e) *Integrating DDD with Microservices Using Web Services*

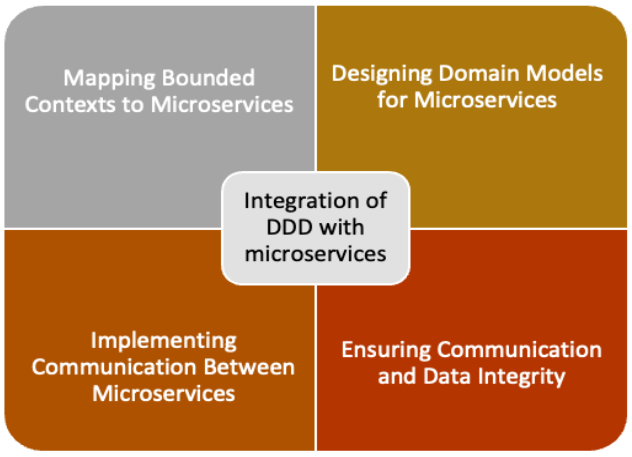
Mapping Bounded Contexts to Microservices
Designing Domain Models for Microservices
Integration of DDD with microservices
Implementing Communication Between Microservices
Ensuring Communication and Data Integrity

Figure 3: Integrating DDD with microservices [12].

f) *Mapping Bounded Contexts to Microservices*

Different activities help to ensure the right designing and management of the microservices architecture. One of such instances is the approach to ensure mapping of bounded contexts to ensure that they can align with the key principles of DDD. Notably, the use of bounded contexts enhances the definition of distinct boundaries in a domain where there is the application of a set of rules, language and model as indicated in Fig.3. The bounded context acts as a representation of cohesive segment of the domain that has its relationships, concepts and constraints [13]. The mapping process therefore has to follow an outlined procedure to enable the settling and management of every model to achieve a desirable outcome as needed. The best process of mapping bounded contexts therefore depends on the following approach:

1) Identification of bounded context: The identification of the bounded context begins, followed by the delineation within the domain. Domain experts are consulted in this case to ensure comprehensive understanding and tackling of different areas within the domain. Some key areas that have to be mapped include elements of understanding business capabilities, organizational units and subdomains. The bounded contexts have to ensure there is a consistency within their boundaries through having related concepts in application.
2) Understand dependencies and interactions: Engaging in a widespread analysis of the dependencies and interactions between the bounded contexts creates an understanding of their relations. Learning about the interrelation of these parties creates a reliable identification of whatever ways they can communicate and collaborate with one another. The interactions are also assessed to understand their frequency and nature, further helping to map their interaction and learning how to design the integration and communication between microservices within these domains [14].
3) Defining Service Boundaries: With the identification of the bounded contexts, one has to define and correspond to the microservice boundaries. The boundaries have to enable identification of functionality and data that pertains to specific context and ensure a high cohesion in the boundaries. Loose coupling also has to be ensured through the identification of contracts, boundaries and APIs that ensure the development of communication and minimization of dependencies between these microservices.
4) Model Domain Entities and Relationships: Each bounded context has to understand and work within the capacity to understand the model domain entities, relationships and aggregates that exist within each categorization. Mapping this engagement ensures that there are domain specific rules, behaviors and concepts that ensure they can conduct their services to achieve provided considerations. Understanding and modelling these relationships encourages the creation of a cohesive and expressive model for each microservice, enhancing the address of their needs at all times.
5) Establish Communication Patterns: Understand the communication within bounded contexts and their framework of interactions. Selection of the right communication framework considering the HTTP/REST APIs has to relate with the nature of

interactions and requirements of every domain [15]. The communication protocols have to be designed to ensure a seamless interaction and association, where they consider the domain and its requirements, enhancing a right level of interoperability and modelling of the microservices to achieve the most desirable end of achieving their projections.
6) Data Consistency and Boundaries: Microservices have to ensure critical handling of data consistency and boundaries to enable their capacity to handle data management in the distributed system with great ease. Boundaries on access and data ownership for every microservice ensures that there is a critical engagement of every microservice to achieve designated elements. Moreover, handling and addressing direct database access by other services ensure that the patterns of bounded context replication and consistency are handled to achieve the data integrity which is needed for every microservice to address their functionalities as desired. In essence, every categorization will structure and achieve meaningful modelling of the data distinctions.
7) Refine and Iterate: Continuous iteration on the bounded contexts will ensure a better adaptation to handle business needs. The use of feedback and evolving requirements and evolving needs will enable the adaptation of the architecture to ensure evolution and achievement of integral advancement of the information to achieve the required insights, creating the right frame of addressing and modelling their functionalities to the objectives and structural elements.

A thorough mapping of the bounded contexts to microservices ensure a possibility to have scalable and maintainable architecture. The microservice architecture will reflect complexities associated with the domain in addressing flexibility, resilience and autonomy in handling requirements of the system in having the required distinctions to a designated appeal.

g) Designing Domain Models for Microservices

Creating domain models that can enable microservices to function well demands a domain-specific approaches that allow them to be cohesive and can apply their core concepts to ensure application and management of every integral need to ensure suitable and applicable development to achieve the required outcomes as indicated in Fig.3. Nonetheless, the domains have to be designed to ensure their core concepts, entities and behavior have the chance to consistently work within the microservice bounded context. The steps to ensure accomplishment and management of this integral approach includes:
1) Understanding the bounded context: Understanding the bounded context of the microservice enhances the capacity to know the capability and applications. Domain experts provide a distinct insight that help to relate with the rules, terminology and requirements that depict an increasingly beneficial way to address the pertinent issue. Nonetheless, the relationships and entities within the microservice also have to be mapped to engage a deeper perspective on their engagement to service.
2) Identify Domain Entities and Aggregates: Domain entities and aggregates represent the domain model of the microservice. Identification of these elements establish a critical point of learning about the microservice domain and addressing pertinent understanding of whatever correlations have to be used to engage the right parties and address desired points of address.
3) Apply the DDD patterns: DDD patterns have to be applied to the complex domain concepts with regard to their value and application [16]. Objects such as aggregates, domain events, value objects, entities and repositories have to be used to ensure domain logic, enhancing their functionality and appeal to greater outcome.
4) Domain Services: Identify the main services within the domains, representative of their operations that have to be addressed within every appeal. Having clear boundaries and responsibilities for every domain service aligns with the microservice domain boundary and objective, creating an even better point of administering valuable insight into managing the domain elements.
5) Relationships and Constraints: Defining the relationships and constraints within domain entities ensure that there is an understanding of the domain's semantics. This further demands the handling of compositions, associations and constrains like uniqueness and cardinality to ensure that there is a properly handled engagement between entities and consistency of data within the microservice domain model. The approach crafts a beneficial way to look into and handle the domain needs.
6) Validate and Iterate: The domain model has to be validated against domain requirements, providing the capacity to address microservice objectives within every distinct classification that is provided. The domain model has to include iterations on evolving demands and feedback in ways that achieve a suitable management of the business needs at all levels [17]. Hence, the model has to enhance on its expressiveness and capacity to address needs of the microservice bounded context.

Implementing Communication Between Microservices

Communication between microservices is an instrumental part of their functionality to enable collaboration within the architecture. Communication patterns and technologies differ, each coming to the point of addressing and working within the capacity to achieve a suitable management of the system requirements. To implement the communication effectively, the following procedures have to be selected:
1) Choose the communication protocol: The right communication protocol has to be selected to assist in addressing interaction between the selected microservices. Some commonly used protocols include RPC, HTTP/REST and messaging Queues.
2) Service interfaces have to be defined to ensure that they are defined and can adhere to the capacity to support one another. The service interfaces work to ensure data exchanges and can handle microservice appeal to individual considerations in attending to and achieving suitable adjustment as demanded.

3) Use the HTTP/REST for enabling synchronous interaction for immediate responses. Asynchronous responses will use messaging queues that will help in defining events that handle important domain issues and can detail changes within the system [18].
4) Implementing the service discovery and registry mechanisms will ensure there is a proper communication between the microservices. The approach relates with the capacity to establish and ensure location and communication without the necessity of hard coding endpoints.
5) Having circuit breakers and retries within the system enhances resilience and fault tolerance to a new level within the microservice. The management of the communication will help to detect and mitigate failed communication helping to ensure that there are better ways to handle network issues that might occur within the microservice domains.
6) Security and Authentication applies to provide a distinct assistance in enabling secure communication between microservices. The use of security also protects sensitive data and marks the chance to ensure that there are encrypted communication platforms that can be used to layer security and aid with working mechanisms as needed. Desirable features of engaging in communication offer a reliable scope and step of addressing the most meaningful indulgence as demanded. Authentication models like API keys, JWT and transport layer security like TLS and SSL will help to provide a great security for the communication and data requirements [19].
7) Monitor and Trace Communication: Monitoring and tracing communication will ensure an instrumental scope and level of addressing performance issues and failures. The use of the right monitoring will help to trace multiple microservices and identify any issues that have to be solved before they affect the proper functionality of the system. Using platforms such as ZipKin and Jaeger will help in providing sustainable engagement and communication to help in addressing remarkable outcomes in achieving suitable outcomes in engagement.

Ensuring Consistency and Data Integrity Across Microservices

Consistency and integrity within microservices help to maintain the reliability of a distributed system within which the microservices operate. Administering the best angle of support and engagement for the system crafts a chance to achieve and ensure remarkable ways of catering for optimal functionality at all levels. Some key ways to ensure the consistency and reliability includes the following approaches:

1) Clear Service Boundaries: Having well defined boundaries for microservice domains helps to address pertinent issues that might result from using the distinct domains. Key boundaries like access rights and data ownership will have the best scope of ensuring that every microservice is key to addressing and managing their capacity to handle business in the right context.
2) Domain Drive Design principles: Employing the DDD principles will ensure that their consistency appeals where they handle requirements and appeal to every recommendation as provided [20].
3) Minimizing the dependencies within the domain models will help to prevent any consistency issues. The use of this approach reduces chances of coupling and makes a step to achieve independent management of microservice domain needs.
4) Implement event-driven architecture: Event driven architecture helps to ensure consistency by ensuring that there are indications on modifications or updates whenever they are conducted. Microservices can relate based on events and ensure that they can consistently work towards achieving these valuable designs at all times.
5) Distributed Transactions: Patterns such as Saga pattern enable the provision of Atomicity, isolation, consistency and durability within the systems. Each of these approaches engage and ensure a critical appeal at the development of much better enshrinements to achieve the required value [21].
6) Monitoring and auditing changes creates the chance to track data and engage in anomaly management to help in addressing pertinent challenges. The best scope and level of achieving sustainability for the information is proactive identification and handling of core inferences that relate to the maintenance service of any upcoming issues. Hence, this approach marks the chance to craft beneficial ways to consider and achieve sustainable engagement of the microservice domains to have consistency [22].

2. Best Practices and Considerations

Various practices stand the chance to ensuring an appropriate integration and use of DDD for microservices using web services. Each of these practices have the chance of enabling and ensuring sustainable advances in managing, addressing and ensuring suitable adjustment to achieve the remarkable benefit of working on the right plane of ensuring sustainable service offering. Some key models that can be used to ensure an instrumental management of the DDD designs for microservices on web services include:

1) Continuous Integration and Deployment: Setting up CI/CD will help to enhance and ensure there are automated ways to compile and package the microservices. Tools like Gitlab and Jenkins have the possibility of ensuring reliably delivery of changes within the handling of microservices without having to alter and engage in differential handling of the platforms. Nonetheless, CI/CD comes with an approach to ensure a continued management of the microservices to ensure that the delivery of various domains is registered and keenly looked into to integrate and ensure firm model handling the traceability element of every version [23]. Nonetheless, having blue-green deployments enable the capacity to handle downtime and work with risks that come during deployment. This helps the microservices to ensure reliability in functions. The continuous monitoring and feedback in the systems also help to achieve and enable considerable achievement in handling consistency of the system.

2) Testing: Different testing modules can be applied to ensure a critical management and handling of the right framework to ensure a remarkable and beneficial path to covering every feature as demanded. Unit tests will help in ensuring individual components live to their functionalities. Integration testing will assist in addressing collaborations and interactions between microservices and ensuring that there is an appropriate communication framework to achieve a better scope of addressing the desired values. Nonetheless, using end to end tests ensure that workflows can be handled to help in achieving and automating elements that simulate user interactions across all angles to ensure that the system behavior and scope of management can be tackled to achieve the most remarkable outcome that is demanded from them [24]. Most to the point, a key test framework that would assist is the use of chaos engineering to identify and remark general development of every step that caters for the system's weaknesses. The engineering will identify weaknesses and help to achieve beneficial management of any underlying weaknesses.
3) Monitoring and Logging: The microservices environment demands a great approach to ensure a secure and instructional management of every step in addressing their demands [25]. Centralized logging will help to collect data and analyze about every microservice and their functionality. Centralized logging can be conducted through Elasticsearch and Logstash to collect information. Having metric collection looks into KPIs within the microservice environment and analyzes their varied levels of performance. This can be conducted with Grafana or Prometheus to understand the performance of every element. Other monitoring techniques like distributed tracing and incident reports are key to the use of the system, aiding a generalized point of advancing critical address of bottlenecks and anomalies that can be defined to craft better outcomes within the system [26].
4) Security Mechanisms for Web Services: Different security approaches can be used to help in addressing imminent threats to the system. Using authentication and authorization for microservices within web services will help to ensure the right access control management and development of key depictions to engage data modelling and handling approach. Nonetheless, working with the transport layer security ensures secure communication and interaction between microservices [27]. Ensuring critical security updates will protect the system and deliver an appropriate amount of engagement to enhance and achieve the right extent of adjusting to challenges experienced within the system.

3. Conclusion

The adoption of DDD for microservices architecture, alongside application of webservices provides a powerful approach to having resilient and scalable software systems. Using DDD enables the use of multiple approaches from domain experts to business experts in mapping and addressing development of core approaches to ensure there are designs that will help them in addressing their needs. Mapping bounded contexts to microservices ensure loose coupling between microservice domains ensuring independence and flexibility in service offering. More to the point, including web services creates a better point for communication and interaction between the microservices in their distributed architecture. Protocols such as HTTP/REST ensure the responsiveness of the architecture and an ease in collaboration to craft and create reliable scope of addressing flexibility and interoperability of the system. Including these changes to the microservices has to factor in the challenges of testing, service coordination and data management needs. These challenges ensure a management and handling of the right appeal, where core identities and definitions are considered as core to depicting the right use of the microservices. Summarily, the use of DDD principles alongside webservices provides a better step to create a powerful framework for modern software. This software can provide key business needs and ensure address of innovative changes that achieve the right to scale and achieve remarkable value development in an adaptive environment. Thus, DDD principles ensure a greater functionality in microservices within the web services environment.

References

[1] V. Khononov, "Learning Domain-Driven Design," O'Reilly Media, Inc., 2021.
[2] S. Millett and N. Tune, "Patterns, principles, and practices of domain-driven design," John Wiley & Sons, 2015.
[3] V. Vernon, "Implementing domain-driven design," Addison-Wesley, 2013.
[4] L. De Lauretis, "From monolithic architecture to microservices architecture," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 93-96, IEEE, Oct. 2019.
[5] S. Li et al., "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review," *Information and Software Technology*, vol. 131, p. 106449, 2021.
[6] C. Surianarayanan, G. Ganapathy, and R. Pethuru, "Essentials of microservices architecture: Paradigms, applications, and techniques," Taylor & Francis, 2019.
[7] V. Raj and R. Sadam, "Evaluation of SOA-based web services and microservices architecture using complexity metrics," *SN Computer Science*, vol. 2, no. 5, p. 374, 2021.
[8] T. Erl, "Service-oriented architecture: analysis and design for services and microservices," Prentice Hall Press, 2016.
[9] J. Ghofrani and D. Lübke, "Challenges of Microservices Architecture: A Survey on the State of the Practice," *ZEUS*, pp. 1-8, 2018.
[10] V. Velepucha and P. Flores, "A survey on microservices architecture: Principles, patterns and migration challenges," *IEEE Access*, 2023.
[11] R. M. Munaf et al., "Microservices architecture: Challenges and proposed conceptual design," in *2019 International Conference on Communication Technologies (ComTech)*, pp. 82-87, IEEE, Mar. 2019.

[12] R. H. Steinegger et al., "Overview of a domain-driven design approach to build microservice-based applications," in *The Thrid Int. Conf. on Advances and Trends in Software Engineering*, Apr. 2017.

[13] C. Zhong et al., "Domain-Driven Design for Microservices: An Evidence-based Investigation," *IEEE Transactions on Software Engineering*, 2024.

[14] S. K. Shivakumar and S. K. Shivakumar, "Modern Web Integration Patterns," in *Modern Web Performance Optimization: Methods, Tools, and Patterns to Speed Up Digital Platforms*, pp. 327-357, 2020.

[15] R. A. Schmidt and M. Thiry, "Microservices identification strategies: A review focused on Model-Driven Engineering and Domain Driven Design approaches," in *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, pp. 1-6, IEEE, Jun. 2020.

[16] H. B. Dinh, "Towards Microservices," doctoral dissertation, University of Applied Sciences, 2023.

[17] M. Waseem et al., "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *Journal of Systems and Software*, vol. 182, p. 111061, 2021.

[18] A. Diepenbrock, F. Rademacher, and S. Sachweh, "An ontology-based approach for domain-driven design of microservice architectures," 2017.

[19] S. Kapferer and O. Zimmermann, "Domain-driven service design: Context modeling, model refactoring and contract generation," in *Service-Oriented Computing: 14th Symposium and Summer School on Service-Oriented Computing, SummerSOC 2020, Crete, Greece, September 13-19, 2020 14*, pp. 189-208, Springer International Publishing, 2020.

[20] M. I. Josélyne et al., "Partitioning microservices: A domain engineering approach," in *Proceedings of the 2018 International Conference on Software Engineering in Africa*, pp. 43-49, May 2018.

[21] C. Schröer and J. Frischkorn, "Decentralized and Microservice-Oriented Data Integration for External Data Sources," in *Innovation Through Information Systems: Volume III: A Collection of Latest Research on Management Issues*, pp. 55-60, Springer International Publishing, 2021.

[22] F. Rademacher et al., "Microservice architecture and model-driven development: Yet singles, soon married (?)," in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, pp. 1-5, May 2018.

[23] H. Vural and M. Koyuncu, "Does domain-driven design lead to finding the optimal modularity of a microservice?," *IEEE Access*, vol. 9, pp. 32721-32733, 2021.

[24] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Migrating to cloud-native architectures using microservices: an experience report," in *Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2015, Taormina, Italy, September 15-17, 2015, Revised Selected Papers 4*, pp. 201-215, Springer International Publishing, 2016.

[25] T. Trad, "Integration testing for enterprise web applications," doctoral dissertation, Politecnico di Torino, 2023.

[26] D. K. Pandiya and N. Charankar, "Optimizing Performance and Scalability in Micro Services with CQRS Design," *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT)*, vol. 13, no. 04, April 2024.

[27] D. K. Pandiya and N. Charankar, "Testing Strategies with Ai for Microservices and Apis," *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT)*, vol. 13, no. 04, April 2024