The Migration of DDD-Oriented Application to CQRS with Event Sourcing Software Architecture

Oleksandr Lytvynov
Faculty of Physics, Electronics and Computer Systems
Oles Honchar Dnipro National University
Dnipro, Ukraine
lytvynov@ffeks.dnu.edu.ua
ORCID: 0000-0001-7660-1353

Dmytro Hruzin
Faculty of Physics, Electronics and Computer Systems
Oles Honchar Dnipro National University
Dnipro, Ukraine
hruzin_dl@ffeks.dnu.edu.ua
ORCID: 0009-0004-8534-2559

Maksym Frolov
Faculty of Physics, Electronics and Computer Systems
Oles Honchar Dnipro National University
Dnipro, Ukraine
frolov_mo@ffeks.dnu.edu.ua
ORCID: 0009-0000-6624-6028

Міграція Інформаційної Системи з Архітектури Domain Driven Design на Архітектуру CQRS та Event Sourcing

Олександр Литвинов
Факультет фізики, електроніки та комп'ютерних систем
Дніпровський національний університет імені Олеся Гончара
Дніпро, Україна
lytvynov@ffeks.dnu.edu.ua
ORCID: 0000-0001-7660-1353

Дмитро Гразін
Факультет фізики, електроніки та комп'ютерних систем
Дніпровський національний університет імені Олеся Гончара
Дніпро, Україна
hruzin_dl@ffeks.dnu.edu.ua
ORCID: 0009-0004-8534-2559

Максим Фролов
Факультет фізики, електроніки та комп'ютерних систем
Дніпровський національний університет імені Олеся Гончара
Дніпро, Україна
frolov_mo@ffeks.dnu.edu.ua
ORCID: 0009-0000-6624-6028

***Анотація*—Стаття розглядає проблему міграції додатків, зокрема тих, що використовують архітектурний підхід Domain-Driven Design, до парадигми Command Query Responsibility Segregation з Event Sourcing. Довго існуючі системи часто стикаються з проблемами, пов'язаними з негнучкою, застарілою архітектурою та залежностями, що призводять до збільшення витрат на обслуговування. У роботі розглядаються переваги DDD та пропонується CQRS як життєздатна альтернатива, з акцентом на покращенні продуктивності та масштабованості. Основною метою роботи є оцінка безпечного шляху міграції проекту з DDD архітектурою на архітектуру CQRS та Event Sourcing, а також визначення дорожньої карти міграції. У статті проводиться експеримент, в якому здійснюється міграція тестового проекту, оцінюються час, зусилля та результати міграції. Методологія дослідження включає оцінку складності за допомогою метрики цикломатичної складності МакКейба та оцінку продуктивності через час виконання методів системи.**

***Abstract*—The work addresses the issue of migrating applications, particularly those following the Domain-Driven Design architecture, to the Command Query Responsibility Segregation paradigm with Event Sourcing. The paper examines the advantages of DDD and proposes CQRS as a viable alternative, focusing on improving productivity and scalability. The main objective of the work is to assess a secure path for migrating a project from DDD architecture to the CQRS and Event Sourcing architecture and to determine the migration roadmap. The experiment is conducted in which migration of a test project is performed, evaluating the time, effort, and results of the migration. The research methodology includes evaluating complexity using McCabe's Cyclomatic Complexity metric and assessing performance through the execution time of system methods. (*Abstract*)**

Keywords—Domain-Driven Design, CQRS, Event Sourcing, Architecture migration (keywords)

Ключові слова: Доменно-орієнтований Дизайн, CQRS, Event Sourcing, Міграція архітектури.

I. INTRODUCTION

In the market, there is a vast number of applications. These are complex information systems that still perform their functions but over time have lost flexibility, and possess outdated architecture, or dependencies. This leads to an increase in the complexity and cost of maintaining such systems. Many of these applications adopt the Domain-Driven Design (DDD) architecture [1][2]. DDD revolves around the concept of bounded contexts [3], which define clear boundaries within a system and promote modular design and clear separation of concerns, allowing different parts of the system to operate independently within their designated contexts. Within these bounded contexts, system entities are identified and organised into aggregates or aggregate roots, which represent clusters of associated objects treated as a single unit for data manipulation and consistency.

One alternative, when it comes to the architecture of complex information systems, is Command Query Responsibility Segregation (CQRS) with Event Sourcing (ES) [4][5]. ES implies the absence of a classical database, and data is stored in the form of events that represent changes to the system's state. Events are stored in an Event Store, which serves as the source of truth for the system. The CQRS architecture provides an innovative methodological approach to optimise command and query processing in applications, contributing to increased productivity, scalability, and modularity of systems.

The advantages of the CQRS with ES architecture compared to DDD [6] include improved performance for read and write requests, as well as better flexibility and scalability, and reduced risk of conflicts when making changes. Another significant advantage is the instant storage of all events, enabling the system's state to be restored to any point in time from its creation to the present.

II. Task definition

For modern information systems, issues of performance, scalability, and ease of software maintenance are crucial. For some systems using DDD architecture, a need arises to store events for monitoring or taking some business solutions, or enhance flexibility, making the CQRS with ES architecture more suitable for the system. However, alongside the advantages, there are practical challenges in its implementation, such as an increase in the number of classes and configuration complexity. Therefore, researching the practical aspects of applying the CQRS architecture for developing a modern information system, analysing and evaluating the pros and cons of this approach, remains relevant.

The objective of this work is to assess a secure path for migrating a complex information system from DDD to the CQRS with ES architecture. Additionally, it involves conducting an experiment to migrate the architecture of a test project and provides an assessment of the time and results of the migration for a test typical project.

III. Literature review

The migration of a complex information system to another architecture is not a simple task and involves several stages. There are various strategies for system replacement, such as Cold Turkey, Chicken Little [7]. Cold Turkey involves rewriting a legacy information system from scratch, while Chicken Little migration assumes small incremental steps until the desired long-term objective is reached. Another approach is Butterfly [8], which focuses on the migration of legacy data in a mission-critical environment. The data migration process is one of the most important and complex steps in system migration. One approach to address this task is the derivation of event logs [9]. This approach involves analysing the system and implementing an events logging module. Events are subsequently used to synchronise data between the old and new systems.

In addition to system replacement strategies, Salvatierra G. [10] considers a number of direct migration approaches, such as Screen Scraping, Sneed, Canfora, COB2WEB, and others.

IV. Materials and methods

Let us describe the migration process of a typical information system from DDD to CQRS with ES architecture.

The first step is to make infrastructural changes, specifically adding the Command Bus, the Event Store, and the Event Bus. After implementing these modules, the controller level and service level (domain) need to be adapted to work with the new modules. At the service level, in accordance with the CQRS architecture, it is assumed that the controller receives a request from an external client, creates a command, and sends it to the Command Bus. Existing controllers need to be updated in the following way. Instead of directly calling domain-level services, controllers responsible for write operations should create corresponding commands and pass them to the Command Bus. The domain services then should be divided and transformed into command handlers and query handlers. Command handlers should subscribe to the Command Bus and process the corresponding commands. The logic for read operations is moved into query handlers. The controllers responsible for performing queries continue to call these methods to retrieve data from query handlers and pass it to the client.

In the next stage, databases for denormalised data views, which are used by query handlers (projections), are described and created. Having ready projections allows creating and testing event handlers, which, upon receiving corresponding events from the event bus, update the projections.

All the previous steps only involved changes in the code and did not affect the system's data. The next step is migrating data from the normalised database to the Event Store and shifting the focus of source-of-truth. During a maintenance break, a database dump is made, and the system is updated to a version which saves events to the Event Store alongside with updating the legacy database. After launching the system, based on the available data in the dump, events of creating existing entities with a timestamp equal to the creation of a record in the database are migrated to the event store. After synchronisation, another maintenance break will be required. Projection databases are filled with data from the Event store using the event replay algorithm, and the system is updated to a version with enabled event handlers that update projections. The last stage of system migration is updating the logic of query handlers to work with projections instead of the legacy database.

V. Experiment and discussion

The Task Tracking System, which is a classic sample project for studying the design and implementation aspects of systems with a complex application domain is chosen as a typical project. The conventional and upgraded solutions are implemented and the source code for the conducted experiment is hosted in an open-source repository on GitHub and is accessible via the link [11].

The development of a typical system took 12 full-time days. The migration lasted another 16 days and increased the number of classes in the business logic from 47 to 213 and required a significant portion of the expended time (Figure 1). At first glance, it may seem that this increase complicates the system. However, the reason for this increase is an improved distribution of responsibilities among system components, which enhances the modularity of the system,

its adaptability to changes, and testability, and ensures its ongoing support and development.

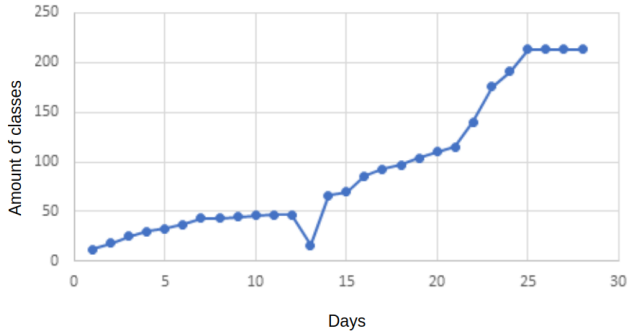

Fig. 1 – Number of classes in the system: Development and migration.

McCabe's Cyclomatic Complexity metric approach [12] was employed to evaluate the code complexity of systems. Despite the increase in the lines of code (3101 vs. 4620) and the addition of new modules, the overall code complexity of the system after migration became even lower (534 vs. 522).

The analysis of the research results indicates a significant impact of implementing CQRS with ES architecture on the system's efficiency. Some methods, which had comparatively high execution time in the conventional system, demonstrated a noticeable reduction of up to 7 times (281 to 43 ms) after the introduction of the architecture. On the other hand, some methods with low execution time nearly doubled (28 to 48 ms).

Acknowledgment

The experiment was conducted on the DBB Software company's [13] proprietary platform, which provided the necessary infrastructure and tools for data collection and analysis.

References

[1] Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software, 2004. ISBN: 978-0321125217.

[2] Vernon V. Implementing Domain-Driven Design, 2013. ISBN: 978-0321834577.

[3] Fowler M. Bounded Context, 2014. URL: https://martinfowler.com/bliki/BoundedContext.html.

[4] Young G. CQRS Documents by Greg Young, 2010. Pages: 50-52. URL: https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf.

[5] Betts D. Dominguez J. Melnik G. Simonazzi F. Subramanian M. Young G. Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure, 2013. ISBN: 978-1621140160.

[6] Kenneth T. Introduction to Domain Driven Design, CQRS and Event Sourcing, 2013. URL: https://www.kenneth-truyers.net/2013/12/05/introduction-to-domain-driven-design-cqrs-and-event-sourcing/.

[7] Brodie M. L. Stonebraker M. Ai S. DARWIN: On the Incremental Migration of Legacy Information Systems, 1995.

[8] Wu B. Lawless D. Bisbal J. O'Sullivan D. The Butterfly Methodology: a gateway-free approach for migrating legacy information systems, 1997. DOI: 10.1109/ICECCS.1997.622311.

[9] Breitmayer M. Arnold L. La Rocca S. Reichert M. Deriving Event Logs from Legacy Software Systems, 2023. DOI: 10.1007/978-3-031-27815-0_30.

[10] Salvatierra G. Mateos C. Crasso M. Legacy System Migration Approaches, 2013. DOI: 10.1109/TLA.2013.6533975.

[11] Frolov M. TaskTrackingSystem repository on GitHub, 2023.

[12] McCabe T. J. A Complexity Measure, 1976. DOI: 10.1109/TSE.1976.233837.

[13] DBB Software's official company site URL: https://dbbsoftware.com/.