Planning and Development Processes for the Information System Based on the Command Query Responsibility Segregation Design Pattern

Krešimir Kavran
InfoDom Ltd.
Andrije Žaje 61, 10000 Zagreb, Croatia
kresimir.kavran@infodom.hr

Maja Pušnik
University of Maribor
Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, 2000 Maribor, Slovenia
maja.pusnik@um.si

Abstract. *The development of a complex information system requires planning and the definition of all business requirements. Business requirements can change over time. Agile methodology enables development teams to adapt to real-time changes and the development of major features according to customer requests. One such complex information system is a platform for the electronic operations of state administration bodies. It contains multiple modules based on the Command Query Responsibility Segregation design pattern. A Croatian software development company created it for the Croatian state administration bodies. This paper describes the development process of creating this information system and explains the maintenance process.*

Keywords. Command Query Responsibility Segregation, Agile, Scrum, Project planning

1 Introduction

When creating a new product or improving an existing one, developers and designers must consider the end-users, their needs, and how they want to use the product. End users care about completing their tasks and achieving their goals, not about how the product was created or the company that created it (Saffer, 2010).

This also affects internal processes in software development companies. Employees must adjust to customer demands and requirements. To address this obstacle, software development companies use different methodologies to plan further development and realize and maintain such plans.

Agile methods, such as Scrum, are currently the most commonly used methodologies in software development (Omonije, 2024). Scrum, in particular, is a framework for developing complex projects and organizing work based on values, principles, and practices that provide a foundation for all members. The Scrum framework offers excellent flexibility and versatility when facing changing constraints (Andrei et al., 2019).

This paper aims to describe the planning and development processes in a Croatian Software development company whose product is an information system based on the Command Query Responsibility Segregation (CQRS) design pattern.

This paper is structured as follows. Section 2 provides background information and related work about Scrum and Command Query Responsibility Segregation design pattern. Section 3 provides information about the information system and its components. Section 4 contains information about the company's planning and development processes. Section 5 includes a discussion. Section 6 concludes the paper.

2 Background and Related Work

This section presents general information about the Command Query Responsibility Segregation design pattern and Scrum. It consists of two parts. The first part describes the Command Query Responsibility Segregation (CQRS) design pattern, including its advantages and disadvantages, while the second provides information about Scrum.

2.1 Command Query Responsibility Segregation

The primary objective of Command Query Responsibility Separation is to distinguish between actions that modify data in the database (referred to as commands) and requests that retrieve data from the database without altering it (referred to as queries) (Overeem et al., 2021).

The benefits of this approach are best observed in distributed environments. Because commands and

queries are separated, it is possible to use different models or define different execution paths for each type. The separation of commands and queries makes each component easier to develop because components do not depend on each other (Kufner & Marik, 2019).

The separation of read and write operations allows the choice of different databases for read and write operations. Developers can select the most performant alternative for queries without sacrificing the benefits of the original (relational) database for state mutation operations. Furthermore, each query can be optimized separately by simultaneously maintaining several different read models. These benefits come with a cost associated with the synchronization of multiple data models and the underlying storage (Debski et al., 2017).

This separation makes each component easier to develop and maintain. While the overall architecture may appear more complex, the individual components become simpler and more manageable, resulting in easier application development (Kufner & Marik, 2019).

Research papers on CQRS are limited in number. The reason for that is the area of application. The CQRS design pattern is intended for use in high-load distributed environments, and as such, it is interesting only to a limited number of solution architects and developers. All published research papers include descriptions of CQRS and additional techniques used in combination with it.

All research papers on CQRS-based information systems include a basic description of the CQRS design pattern and descriptions of other techniques used.

Published research papers describe various elements, implementations, advantages, and disadvantages of CQRS.

The authors (Diakov et al., 2019) compare the CQRS design pattern with traditional CRUD systems, and (Weerakoon & Kumara, 2018) compare CQRS with monolithic architecture, focusing on eventual consistency and its effects.

Research papers (Debski et al., 2017) and (Overeem et al., 2017) focus on event sourcing and describe the benefits and challenges of using event sourcing in CQRS applications. While (Lima et al., 2021) propose improvements to event sourcing in CQRS-based applications.

The authors (Erb & Kargl, 2014) describe how to use the CQRS pattern in event-driven architectures, while (Kufner & Marik, 2019) explain how to implement the CQRS design pattern in REST APIs.

The research paper (Kabbedijk et al., 2012) introduces techniques to enhance the CQRS design pattern, such as aggregating roots and snapshotting to increase variability.

Finally, (Rajković et al., 2013) present improvements in the performance of existing information systems by using two databases, each for read and write operations, along with a method for data synchronization between them.

2.2 Scrum

In 2001, seventeen software development professionals published the Agile Manifesto. The Agile Manifesto focuses development on four core values. The Agile Manifesto also contains twelve additional principles to ensure user satisfaction by continuously delivering valuable, simple, high-quality software (Itzik & Roy, 2023).

In Agile, teams are self-organized, allowing teams to adjust to any problem and choose the best approach for solving problems. The definition of roles in agile software development depends on the development method and the organization's size (Bomström et al., 2023).

Scrum is a management and control process that cuts through complexity to focus on building software that meets business needs (Schwaber & Beedle, 2002).

The word "Scrum" originated from the sport of rugby, where a Scrum is a formation that enables quick adoption of strategies by team members, with every player playing a specific role (Zayat & Senvar, 2020).

Key elements of Scrum are:

- *Product Backlog* is a prioritized list of all product requirements. It describes everything the system should include regarding functionality, features, and technology.
- *Product Owner* is responsible for prioritization of the backlog. The Product Owner decides the order in which things are built (Schwaber & Beedle, 2002).
- *Scrum Teams* are small, cross-functional teams that perform all development. They are also referred to as development teams.
- *Sprints* are development iterations lasting between one and four weeks. Every Sprint must finish with an increment of the system and new product functionality.
- *Sprint Backlog* is a collection of items taken at the beginning of a Sprint by the development team that can be completed during the Sprint and turned into an increment of the system.
- *Scrum Master* is a management representative who enforces Scrum practices and helps the development team to make decisions or acquire resources as needed (Schwaber & Beedle, 2002).
- *Daily Scrum* is a daily meeting held by the Scrum Team to review progress and identify problems.
- *Sprint Review Meeting* is a meeting between the Scrum Team and the management held at the end of every Sprint. It is used to inspect the product increment that has been built.

3 The Information System

The chosen information system is used for the electronic operations of state administration bodies. It is intended for state administration bodies, local and regional self-government units, public authority bodies, and all enterprises that provide public services to citizens and businesses. Croatian software company created and maintains it, and Croatian state administration bodies use it.

The information system is a modular microservice-based information system, with its core components built around the CQRS design pattern. It is composed of many modules. The most used modules of the information system are:

- primary web interface,
- secondary web interface,
- modules for integration with government services,
- plugin for e-mail clients,
- integration service.

Regardless of the modules used, all modules employ the same core components, ensuring their basic functioning is consistent, and they all utilize the CQRS design pattern.

The primary user interface is intended for employees of state administration bodies. This web interface is what most end users use in everyday business. It is the most complex module and offers access to all functionalities of the information system. It utilizes role-based access control, allowing employees to access specific functionalities based on their assigned roles.

The secondary web user interface is intended for citizens and businesses to use. Its goal is to enable more accessible communication with state administration bodies, simplify communication, facilitate the submission of requests to state administration bodies, and allow citizens to monitor the progress of their inquiries. A secondary user interface is not obligatory for the correct work of the information system.

Modules for integration with government services enable the retrieval of data, such as information about citizens or companies, from government services. Using these services, the information system can always contain up-to-date information without requiring employees to manually update the records. Each government service has a separate module in the information system. These modules are not standalone solutions and must be incorporated with other modules.

An elective option for the information system is a plugin for e-mail clients, which enables inputting data directly from the e-mail client without using the primary web interface. A plugin for e-mail clients was created to allow employees to input citizens' e-mails, contact information, and data into the information system and automatically generate and send confirmation e-mails.

Depending on the employee's work position, a primary web interface may not be needed for the employee to complete tasks. Some users are tasked with communicating with citizens, and by using this plugin, they can complete their tasks without needing to use the primary web interface, thereby reducing the load on the web interface.

Integration service is a module that enables the exchange of data with other information systems. If a customer has another application or information system from a different manufacturer, they can utilize this integration service to establish a connection between the two solutions, thereby eliminating potential data duplication across multiple information systems. Integration service does not offer all the functionalities of the primary web interface. It provides most read-and-write operations. Operations that are not supported include those that could affect the entire information system's functionality, such as configuration properties and limitations imposed by legal regulations.

The information system was developed as a modular microservice system. Fig. 1 shows the simplified architecture of the information system for end users (the central part of the information system).

All end users' requests, regardless of how they are made, are sent to the Representational State Transfer Application Programming Interface (REST API) for data manipulation (data retrieval, inserting new data, or updating existing data).

Web user interfaces use separate controllers from other modules. Most controllers in the REST API are generated at startup, depending on the available queries and commands. This simplifies adding new queries and commands because no configuration is needed to use them. Some specific controllers are not generated but are written by developers to enable the execution of particular tasks. Due to this level of customization, only web user interfaces created by the company can utilize them.

E-mail client plugin and other modules use separate controllers. These controllers are written by developers and are not generated at the startup. These controllers form an integration service that enables other information systems to exchange data with this information system.

This means that only actions permitted by the company are allowed for use with other information systems. Because other information system modules perform specific tasks, unlike web user interfaces, separate controllers were made. These modules do not require all available operations, such as web user interfaces.

Controllers are responsible for accepting user requests and determining whether the request should be forwarded to other components or if the controller can process the request itself without needing to query the database and send the response to the web interface.

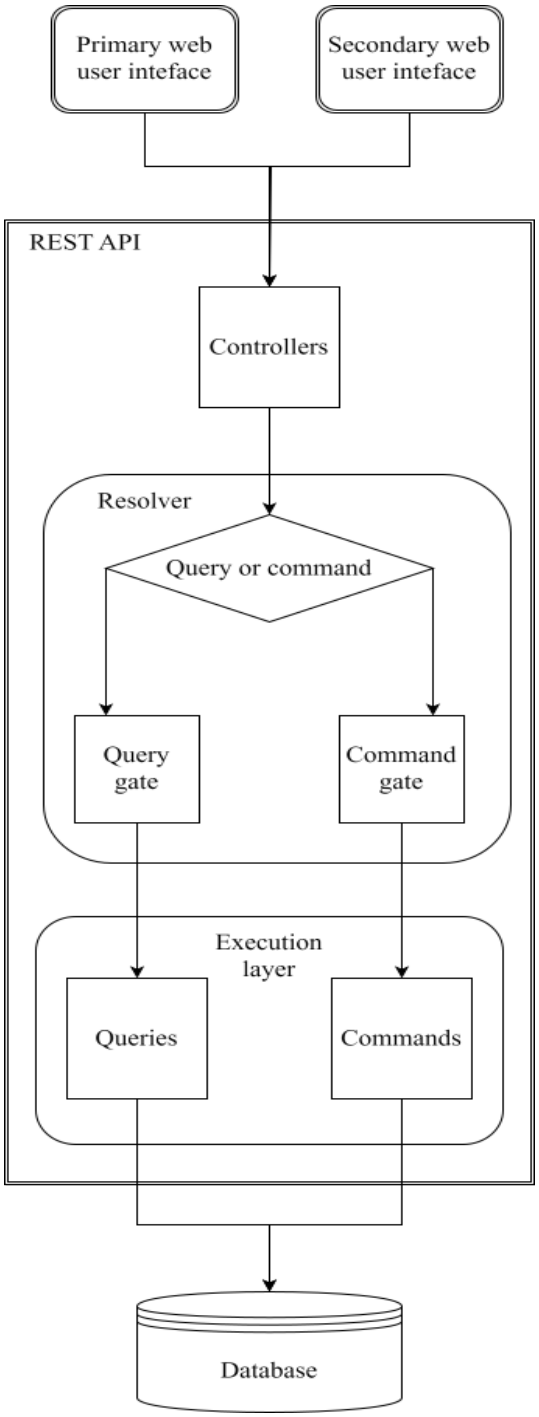

Figure 1. Information system architecture overview

If access to the database is needed, whether for a query or command, the request is forwarded to the resolver. The resolver determines whether a query or command needs to be executed to process the request. The request is sent to the query gate if it is a query. Otherwise, it is sent to the command gate.

Each gate executes a query or command in the execution layer. Depending on what component was called, the result can be a data set (the result of a query) or information on whether modifications to the data were successful (the result of a command). The list of queries and commands is not static. It is automatically generated at the start time, and no configuration is needed to add a new query or command. This enables a quick and easy way to add new queries and commands or modify existing ones.

The execution layer executes queries and commands on the given database. It can be configured to use one or more databases. The default configuration utilizes a single database, but it also supports multiple databases. For example, it can be configured to use two databases. In that case, one database can be used for queries (read) and a second for commands (write). In the case of multiple databases, it is necessary to ensure data synchronization between databases.

4 Planning and Development Processes

The company uses the Agile method Scrum for planning and development. To simplify the entire process, the company uses the tool Azure DevOps. This section contains descriptions of Azure DevOps and planning and development processes.

4.1 DevOps

The acronym DevOps comes from the concept of development and operations. These are related to the creation of software products. Its challenges include the adequacy of applying techniques and software development processes that align with all necessary operations to deliver a quality product. In this sense, a relationship between quality, costs, operations, time, and activities is essential and must be related to the participation of all software development team members (Bento et al., 2023).

DevOps is a set of practices that integrate software development and IT operation teams by working together across the entire software development life cycle (Dileepkumar & Mathew, 2021).

Azure DevOps is a professional tool used by companies specializing in software development (Bento et al., 2023). It is widespread worldwide and has a large amount of documentation and working methods. Azure DevOps enables organizations to manage their end-to-end software development and deployment processes effectively and efficiently (Dileepkumar & Mathew, 2021).

Azure DevOps is composed of several key components that form its backbone. These include

- Azure Repos, which manages source code repositories, Azure Pipelines for continuous integration and delivery,
- Azure Boards for agile project management,
- Azure Artifacts for package management.

These components work together to streamline the software development process, thereby enhancing team collaboration and productivity (Borra, 2024).

Azure Boards is a versatile and adaptable tool for organizing and overseeing work across teams. It facilitates agile project management functions, including backlog management, sprint planning, task tracking, and visual boards. With customizable features such as work item configuration, burndown charts, and seamless integration with other Azure DevOps services, Azure Boards ensures that teams maintain visibility and transparency, which are crucial for alignment and productivity (Borra, 2024).

Azure Repos provides Git repositories designed for efficient code version control, enabling seamless collaboration among teams. Regardless of project scope, teams benefit from features such as branching, pull requests, code reviews, and branch policies, ensuring code quality, consistency, and collaboration (Borra, 2024).

Azure Pipelines emerges as a robust continuous integration and continuous delivery (CI/CD) service, automating the build, test, and deployment processes. It accommodates various programming languages, platforms, and deployment targets, seamlessly empowering teams to deploy applications to any cloud or on-premises environment. With features such as multi-stage pipelines, parallel execution, and integration with third-party tools, Azure Pipelines enables teams to deliver software updates confidently and efficiently (Borra, 2024).

Azure DevOps is used in the selected company for planning, development, building new versions, testing, and deployment. The project is based on .NET technology, and the logical choice was to use the most compatible tool.

4.2 Planning process

The chosen information system is designed to support state administration bodies in their operations. For this reason, multiple sources influence the planning process and future development. These sources offer ideas or requests for improving the information system and defining new functionalities. The source can be an employee, a customer, a legal regulation, or a third party (for integration services). This means that the need for new functionalities rises during the year, and because the company uses Scrum, that is not a problem.

Azure Board enables the definition of new functionalities and agile ways of work management. The company uses Azure Board to log incoming requests or ideas as work items regardless of the source. Also, it enables tracking completed work.

For this reason and planning, it is necessary to group all related functionalities. As mentioned before, the company uses Azure. The Azure Board enforces grouping all work items into groups called Epics and Features.

The company uses group Epic to group work items per project or, in the case of complex projects, into multiple Epic groups. In the case of the chosen information system, the company uses the following Epic groups:

- one epic group for main functionalities and minor modules of the information system,
- one epic group for architectural improvements,
- three epic groups for the three most significant modules and improvements to the information system,
- one epic group for deployment and implementation of solutions in the customer's environments for core components and all modules except one module,
- one epic group for deployment and implementation of the most complex module.

These groups were created to simplify the process and enable easier tracking of completed work, rather than placing everything in one group. Creating a new epic group is allowed only for senior employees and only for legitimate business reasons.

Some of the properties for epic groups are group name, description, start date, and target (end) date. Because this is a long-term project, these epic groups do not have a defined target date. The information system is constantly updated with new functionalities and improvements.

Feature groups are part of an epic group, representing specific information system features. A feature group is a core component, a module component, an improvement requested by the customer, and other information system features. When this paper was written, there were 141 feature groups distributed in seven epic groups.

Feature groups have the same properties as epic groups, like group name, description, start date, and target (end) date. All feature groups have a target date except those used for core components.

Azure Boards (a part of Azure DevOps) offers a graphical representation called a feature timeline of all feature groups for easier planning and prioritizing work. It uses a Gantt chart for a graphical representation of feature groups.

Feature groups contain User Stories and Bugs. A User Story is a work item that includes a description of a new feature or improvement of an existing feature of the information system. A Bug describes a found or reported bug in the information system. User Stories and Bugs must include all relevant information for developers to create new features, improve existing features, or fix bugs in the information system. They represent assignments for the developers.

Each user story and bug must contain a Task. A task represents specific actions that a developer must do to fulfill the assignment. A user story and a bug must contain at least one task.

User stories and bugs can contain issues besides the tasks. An issue is added later in the development if an error is found in the solution during the testing, or if

something is not done as described in the user story or bug. The issue can be added to the user story or bug only if the user story or bug is still in the sprint backlog. A new Bug must be created if the original user story or bug is not in the sprint backlog.

In the company, only senior employees can create epic and feature items (groups). Everyone can create user stories, bugs, tasks, and issues, regardless of their position or role.

All employees are divided into two groups. These groups are:

- business group (business analysts, project managers, product owner, implementers, testers),
- technical group (developers, solution architects, project technical managers, scrum master).

Business group members are responsible for communicating with the customers, configuring the features of the information system, and testing new versions. The business group does not have permission to modify the source code.

Technical group members are responsible for developing new features or improving the existing features of the information system. The technical group has permissions to access and modify the source code.

The information system is not available to everyone. It can only be obtained by directly contacting the company and is available only for specific clients. The information system is designed for use by state administration bodies and state agencies. When buying the information system, a contract is made between the customer and the company. The contract lists all details about the installation, maintenance, and deployment of new versions. A dedicated contract is made if a customer wants a specific new functionality.

The company assigns a project manager and a project technical manager to every customer. A project manager is tasked with communicating with the customer's representative about the functionalities of the information system and the deployment of new versions (if deployment and maintenance are listed in the contract).

The project technical manager is responsible for assisting the project manager with technical issues and, if necessary, communicating with the customer's representative about maintenance and technical issues (if that is listed in the contract).

Business group members are responsible for creating new work items for the developers. Depending on the source of the work item, a work item can have a target date or not. If the source for the work item is a customer, there is often a target date.

For new functionalities requested by the customer, a new dedicated contract with the customer must be created and signed. If the customer insists on a specific deployment date for the new version with the new functionality, and a target date is specified in the contract, new work items will also have a target date. Before the target date, a new version must be deployed. On the target date, a new version must be given to the customer for use.

There is no need for a separate contract if the customer reports a bug. If it is a critical bug, a solution will be deployed as soon as possible. If it is a minor bug, the solution will be deployed on the scheduled date (if continuous deployment of the new version is specified in the contract).

The deployment of new versions varies per customer. Some customers prefer monthly deployments, while others opt for a new version deployment once a year.

Business group members must decide if existing feature groups can be used for new work items when planning future work. If the existing feature group fits the purpose for new work items, the employee creates new user stories and bugs inside that feature group.

If existing feature groups are not suited, senior members must create a new feature group and decide on the group name and its properties. One of the important properties is the target date because this property influences the feature timeline and deployments. Also, it influences the prioritization of work items.

Depending on the complexity of the new functionality, an employee creates one or more user stories. A user story must be completed within a sprint. If that requirement can not be fulfilled, multiple user stories are made.

Feature groups that do not have a target date are dedicated to user stories for functionalities that are nice to have but are not requested by the customer, or for functionalities for which a release date has not been decided. Also, feature groups in the epic item from architectural improvements do not have a target date. These groups are dedicated to technical group members and user stories with tasks for optimizing or improving core components that affect the entire information system.

Business group members have meetings every two weeks to prioritize work items (user stories and bugs). These meetings occur before sprint planning for the upcoming sprint. The prioritization is done according to the feature timeline (current contracts) and by the significance to the information system.

When a work item (user story or bug) enters a sprint, responsible business group members must be available to the developers for additional information regarding the work item (if necessary).

After a new version of the information system is built and deployed on the company servers, testers conduct tests on the information system. If an issue is found and a user story or bug is still present in the sprint, the tester creates an issue and notifies the responsible developer, whose task is to resolve the issue as soon as possible. If a tester finds an issue whose related user story or bug is not in the sprint, he/she must create a new work item (bug).

If a new version does not have critical bugs, it becomes a release candidate. The project manager's task is to decide if the new version will be deployed on

the customer's servers. This also depends on the signed contract and deployment plan listed in the contract.

Project managers must contact customer representatives if a new version needs to be deployed. After agreeing on the time and date of the deployment, project managers must communicate the date and time to the project technical manager. The project technical manager is responsible for the deployment of the new version. If the project technical manager is unavailable to perform the deployment, senior members of the technical group must decide who will perform the deployment instead.

Deployments can be automated or manual. Deployments on the company test servers are automated. The deployment method varies per customer due to infrastructure, business, and security restrictions. An example of manual deployment is the use of an information system within the customer's local network, without internet access.

4.3 Development process

The technical group is responsible for development. The group consists of developers and solution architects, some of whom hold the role of project technical manager. The group is also referred to as the development team.

At the time of writing this paper, the technical group consisted of

- three junior .NET developers,
- three mid .NET developers,
- two senior .NET developers,
- three SQL developers,
- two solution architects,
- scrum master/solution architect.

Developers are responsible for developing new features, improving existing features, and fixing bugs. Solution architects are responsible for the architecture of the entire information system and deciding on the best approach for implementing new features or improving core components. The development team has three solution architects, but not all three are in charge simultaneously. One solution architect, who also serves as a Scrum Master, is the lead solution architect. The remaining two solution architects support the lead solution architect, and each is responsible for a different area. One is dedicated to databases, while the other is dedicated to programming. In the absence of a lead solution architect, one of the remaining two solution architects assumes the role of lead solution architect.

Technical group members are shielded from outside influence and are a self-organizing team. The exceptions are project technical managers who only communicate with the customer representative if necessary.

The development teams' work is organized into sprints. One sprint lasts two weeks. The exception is the month of August, when the sprint lasts the entire month because most developers are on vacation. In August, the development team works at half capacity due to vacations and the shift of developers returning from vacation and those just going on vacation.

Before every sprint, the development team organizes a meeting called sprint planning. That meeting is held a day or a couple of days before the start of a new sprint, depending on the available time of the development team members. All development team members are required to participate in the meeting. Also, the product owner is obligated to be present at the meeting. Sprint planning is led by the lead solution architect or, in the case of absence, one of the two remaining solution architects.

At the beginning of sprint planning, all unfinished items from the previous sprint are transferred to the new sprint. After that, new items from the product backlog are added to the new sprint according to the priority established by the business group members. New items are added until 70% of sprint capacity is achieved.

In Azure DevOps, every developer has an assigned daily capacity from which a total capacity for the sprint is calculated. Developers' daily capacity depends on their knowledge and experience. It varies from three hours (for junior developers) to six hours (for senior developers). The remaining working hours are allocated for breaks, code reviews, learning, and knowledge transfer.

The 70% limit was chosen based on previous experiences. During the sprint, unforeseen circumstances could result in the absence of one or more developers or solution architects. The reason for absence could be illness, family problems, or specific emergency problems. Also, a critical bug could be found during the sprint, which requires immediate action. These bugs are added later during the sprint in the sprint backlog and also affect the team's capacity.

The installation of the information system on the servers of the new customer or the installation of new modules on the servers of existing customers are user stories (work items) that can be added to the sprint backlog during sprint planning. Reasons for this are contracts that specify fixed dates for installation and delivery, and because the initial installation or installation of additional modules requires the installation of support software on customer servers.

Deployments of new versions are not user stories (work items) because the deployment process is simplified, and the date for deployment is agreed upon when a new stable version is created (unless the date is also listed in the contract or if a version contains a feature request by the customer). For deployment of upgrades, it is unnecessary to install any additional support software (it is already installed on the servers). This is also one of the reasons for the limit of 70% capacity.

Work items are not immediately assigned to each developer during sprint planning. Instead, each

developer takes over a new work item once the previous one is completed. Junior developers can choose which work item they want to work on next. This is permitted due to their limited experience and knowledge. Junior developers take tasks they already know how to complete or tasks for learning purposes. All other developers and solution architects must take work items according to priority. The exceptions are sensitive tasks (modification of core components or task complexity). These work items are immediately assigned to the senior developer or solution architect.

A daily meeting is held during the sprint at the beginning of each work day. Members of both groups are present at the daily meetings. At these meetings, technical group members must report on their activities from the previous working day and announce their tasks for the current day. At the end of the meeting, business group members are allowed to announce any new information they have obtained during the previous day (if necessary for the development). Daily meetings can last between five and twenty minutes, depending on the number of present technical group members.

The development team holds weekly meetings called development backlog meetings, during which technical topics are discussed. The topics of these meetings include the presentation of changes to core components, discussion on the best way to implement a specific new feature, assisting another developer in solving a particular problem they encountered during development, and other related topics. The duration of this meeting is approximately one hour. Development backlog meetings are not always held every week. It depends on the number of available developer team members and if there is a suitable topic for the meeting.

A sprint review and retrospective are held at the end of the sprint. Members of both groups are present at these meetings. The purpose of a sprint review and retrospective is to present new features or significant improvements to existing features (if any have been introduced) to all employees working on the project, and to discuss problems encountered during the sprint and possible solutions to improve future sprints.

Additionally, during these meetings, a discussion on metrics is held. The most used metrics are velocity, sprint progress, burndown chart, pipeline pass rate, and deployment pass rate. Metric results are discussed at the meeting to determine possible problems and bottlenecks. Especially if a negative trend is identified, this discussion enables every member to express their opinion and fosters open communication. It also creates an opportunity for every member to propose a solution for improving planning and development processes.

5 Discussion

The previous section describes planning and development processes separately, but it is necessary to view these processes as a whole. Members of both groups must work together to improve and maintain the information system. Fig. 2 illustrates a simplified flow diagram for both processes, along with the corresponding links between them.

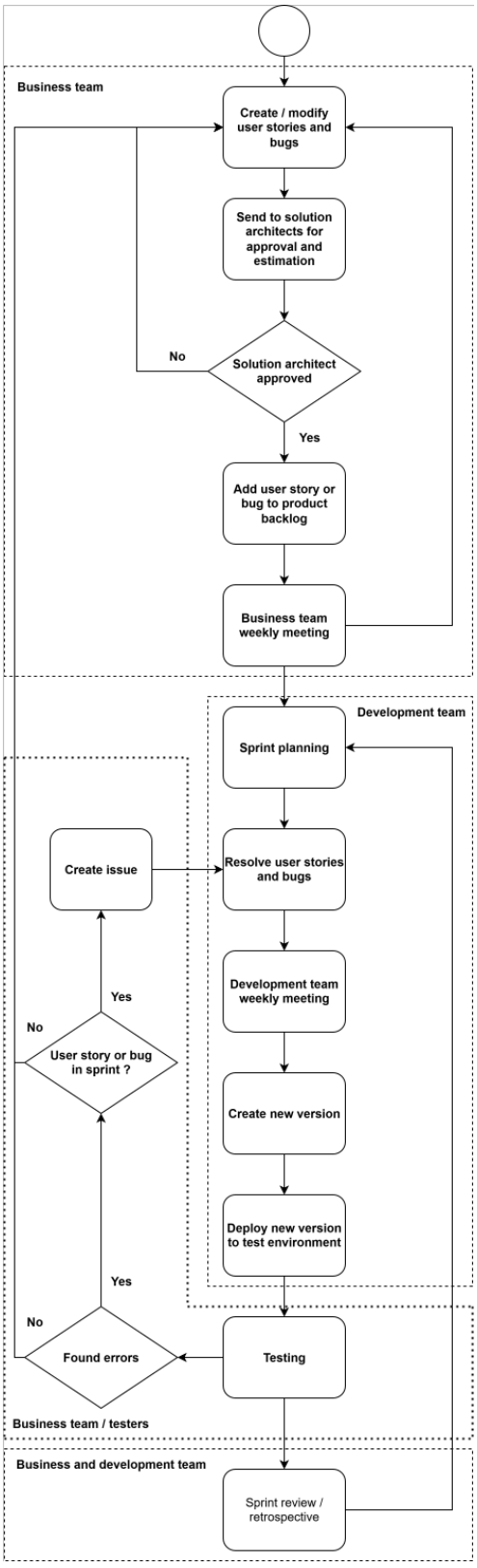

Figure 2. Planning and development processes

The company's solution architects, project managers, and product owner cooperate when defining new work items (user stories and bugs). After the business team member creates a work item, the solution architect reads what was written and leaves a comment. These comments are helpful because they enable the business team member to correct any mistakes or provide additional information before a work item is sent to the developer. It also helps developers to better understand what is asked of them.

The entire process begins with a discussion between business team members about new functionalities or improvements, regardless of whether the customer has requested it or the idea originated from an employee. Anyone can propose ideas for improving the information system. Sometimes, changes to the information system are necessary due to updates in legal regulations.

After creating a work item, if the employee is satisfied with the content, it is sent to the solution architects for review. The original author of the work item becomes the owner of the user story or bug. If the work item is well-written and contains all the necessary information, a solution architect assesses the effort required for the developer to complete all tasks associated with that work item.

If the solution architect is not satisfied with the description in the work item or if relevant information is missing, the solution architect leaves a comment and asks for an amendment or makes modifications to the work item. Regardless of what a solution architect does, a work item is returned to the owner.

If changes were made to the work item, the owner's task is to assess the changes and send a review to the solution architect. If a comment (review of the solution architect) was left, the owner must decide whether and what changes will be made to the work item and send the modified work item to the solution architect. This process repeats until the owner and solution architects are satisfied with the contents of a work item.

This has been shown to be a best practice for creating new work items, as it helps both employees understand each other's positions better and shortens development time when necessary information is provided in the work item. The disadvantage is that it takes longer to write work items and creates additional assignments for solution architects. However, compared to previous practices where only business team members wrote work items, it demonstrates better communication between employees and increased user satisfaction. It is easier for a developer to program something when the description and all resources are in the work item. This means fewer bugs in the newly added feature, which results in a better user experience.

After an estimate is given, the work item enters the product backlog at the end (the last item in the product backlog). It stays at the end of the product backlog until the prioritization meeting. At this meeting, all members of the business teams discuss all new work items and decide on a priority. The priority of work items is determined based on the feature timeline (Gantt chart), the agreed-upon deployment dates, and members' options. At that time, work items can change position in the product backlog.

At that point, business team members are done with their work on the discussed work item, and the technical team members (development team) consider these work items during the next sprint planning. At sprint planning, new work items are added to the new sprint until the capacity reaches the 70% limit of the maximum capacity. New items are added according to the previously determined priority.

Most work items are discussed during sprint planning before being added to a new sprint backlog. Some work items are not addressed because they are simple and have straightforward solutions. This discussion is the final point at which work items can be discussed before proceeding to development.

In practice, this discussion has proven beneficial because it allows other development team members to point out possible mistakes or business rule violations. Additionally, it is beneficial for junior developers to observe the thought process of evaluating work items and develop an understanding of what is essential for effective work items. It enables junior developers to advance their knowledge and learn how to estimate the effort needed to resolve work items.

During sprint planning, the development team can reject a work item after discussion if they determine possible mistakes, limitations, or conflicts with other work items. The rejected work items are returned to the business team and are not added to the sprint backlog. The rejected work items can be returned to the development team for the next sprint planning after modification of the description, depending resources or additional clarification.

After sprint planning, when a new sprint begins, the development team starts working on work items in the sprint backlog. During this time, business team members are available to answer any questions that may arise during development.

During the sprint, the business team works on work items for the next sprint and tests and evaluates work items that have already been completed from the previous or current sprint (if a new version is built during the sprint). Additionally, the business team must maintain contact with customers to understand their needs and assist them in resolving any issues that arise during the use of the information system. The task of the business team is to shield developers from outside influence and enable them to focus on solving assigned work items.

The business team also holds daily meetings to discuss its plans. The development team does not participate in these meetings. At these meetings, members discuss feature groups, deadlines, and future deployment dates per customer.

The development team holds daily meetings where every member is required to report on their previous workday and outline their plan for the current workday.

It is beneficial for senior developers and solution architects to know what the junior developers are working on because it enables them to predict if or when they would need assistance. It helps senior developers and solution architects plan their daily obligations.

For junior developers, daily meetings are important to see what could be achieved if they choose to stay in the IT business domain and invest time in learning and code analysis of solutions made by the senior developers.

Every week, the development team has a development backlog meeting. These meetings are used to present significant changes to the core components, address questions and problems for junior developers, and discuss future changes to the information system's architecture. As mentioned before, there is one epic group for architectural improvements. This group is reserved for the development team for features and work items related to the information system's architecture. In this group, developers create work items for themself. These work items address problems that are not known to the business team. The business team lacks insight into the information system's background process and core components.

Developers are responsible for creating work items and further improving core components. Senior developers and solution architects propose ideas at development backlog meetings, and the entire team discusses them. If the proposed changes to the information system would provide benefits, a selected senior developer or solution architect creates a work item. Also, the development team proposes a priority for the work items.

Work items created by developers follow the same procedure as all other work items. Work items are sent to the business team. The business team discusses the work items and sends feedback to the development team. The business team assigns priority to these work items based on the development team's proposition, and most of the time, it is the same as proposed.

At sprint planning, at least one work item created by the development team is added to the sprint backlog, and a senior developer is immediately assigned. The assigned developer is usually the same developer who created the work item. When a new version is built, testing completed work is done by other senior developers or solution architects. Testers do not test these solutions.

During the sprint, one or more new versions of the information system are built. At least one version of the information system must be built at the end of the sprint.

When a new version is created, it is immediately deployed on company servers, and the business team is notified. Testers and project managers are responsible for testing the latest version and analyzing completed work. Completed work items can be returned to the development team for necessary changes. These requests for changes can be made in the form of an issue, a completely new user story, or a bug.

New versions should be presented at the end of the sprint at the sprint review and retrospective meeting, but in practice, that is not the case. Depending on the impact of the changes, that part of the entire process is sometimes skipped.

The described process works best for these teams and their project. This is evident in the company's overall income and customer satisfaction.

6 Conclusion

This paper presented the planning and development process for one of the projects in the Croatian software development company. The paper also describes the chosen information system to provide full context.

The aim of this paper is to describe how the Scrum framework was adopted for the selected project, the impact on the team, and the best practices that emerged, highlighting how this implementation enabled agility in response to quick changes. This implementation demonstrates how the Scrum framework can be applied in practice and customized for each company, project, and team.

The presented processes show the current state of the project. This project is an ongoing project and will continue to be until a new iteration of the information system is started (the new information system started from scratch in the newer technologies) or the customers find a better product.

The processes have possibilities for improvement. However, whether the improvement will be accepted or not will be shown if further analysis and follow-up research are conducted.

References

Andrei, B.-A., Casu-Pop, A.-C., Gheorghe, S.-C., & Boianggiu, C.-A. (2019). A study on using waterfall and agile methods in software project management. *Journal of Information Systems & Operations Management*, 13(1), 125-135

Bento, A. C., Delgado D. A. C., & Camacho-Leon, S. (2023). Experimental Survey Results on Azure.DevOps Application for Management Student's Projects. *2023 Future of Educational Innovation-Workshop Series Data in Action*, 1-5

Bomström, H., Kelanti, M., Annanperä, E., Liukkunen, K., Kilamo, T., Sievi-Korte, O., & Systä, K. (2023). Information needs and presentation in agile software development. *Information and Software Technology*, 162, 107265

Borra, P. (2024). Maximizing Efficiency and Collaboration with Microsoft Azure DevOps. *International Journal of Advanced Research in Science, Communication and Technology*, 4(2), 556-562

Debski, A., Szczepanik B., Malawski, M., Spahr S., & Muthig, D. (2017). A Scalable, Reactive Architecture for Cloud Applications. *IEEE Software*, 35(2), 62-71

Diakov, S. O., Zubrei, T. E., & Samoidiuk, A. S. (2019). Application of Event Sourcing and CQRS Patterns in Distributed Systems. *Adaptive Systems of Automatic Control*, 1(34), 16-22

Dileepkumar, S. R., & Mathew, J. (2021). Optimize Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft .NET environment using different techniques and practices. *IOP Conference Series: Materials Science and Engineering*, 1085(1), 012027

Erb, B., & Kargl, F. (2014). Combining Discrete Event Simulations and Event Sourcing. *Seventh International Conference on Simulation Tools and Techniques*, 51-55

Itzik, D., & Roy, G. (2023). Does agile methodology fit all characteristics of software projects? Review and analysis. *Empirical Software Engineering*, 28, 105

Kabbedijk, J., Jansen, S., & Brinkkemper, S. (2012). A Case Study of the Variability Consequences of the CQRS Pattern in Online Business Software. *17th European Conference on Pattern Languages of Programs (EuroPLoP '12)*, 2, 1-10

Kufner, J., & Marik, R. (2019). Restful State Machines and SQL Database. *IEEE Access*, 7, 144603-144617

Lima, S., Correia, J., Araujo, F., & Cardoso, J. (2021). Improving observability in Event Sourcing Systems. *Journal of Systems and Software*, 181, 111015

Omonije, A. (2024). Agile Methodology: A Comprehensive Impact on Modern Business Operations. *International Journal of Science and Research (IJSR)*, 13(2), 132-138

Overeem, M., Spoor, M., & Jansen, S. (2017). The Dark Side of Event Sourcing: Managing Data Conversion. *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 193-204

Overeem, M., Spoor, M., Jansen, S., & Brinkkemper, S. (2021). An empirical characterization of event sourced systems and their schema evolution - Lessons from industry. *Journal of Systems and Software*, 178, 110970

Rajković, P., Janković, D., & Milenković, A. (2013). Using CQRS Pattern for Improving Performances in Medical Information Systems. *6th Balkan Conference in Informatics (BCI 2013)*, 86-91

Saffer, D. (2010). *Designing for Interaction: Creating Innovative Applications and Devices, Second Edition*. New Riders.

Schwaber, K., & Beedle, M. (2002). *Agile Software Development with Scrum*. Prentice Hall

Weerakoon, W., & Kumara, B. T. G. S. (2018). Data Handling and Maintaining Data Consistency in Scalable Replicated Micro-Services. *11th International Research Conference General Sir John Kotelawala Defence University*, 281-286

Zayat, W., & Senvar, O. (2020). Framework Study for Agile Software Development Via Scrum and Kanban. *International Journal of Innovation and Technology Management*, 17(04), 2030002