Software Reengineering with Object-Oriented n-Tier Architecture: Case of Desktop-to-Web Transformation

*A. Stojkov, **Lj. Kazi and ***M. Blažić
University of Novi Sad, Technical Faculty "Mihajlo Pupin", Zrenjanin, Republic of Serbia
sahskica.s@gmail.com, ljubica.kazi@gmail.com, markoblazic93@gmail.com

Abstract – This paper presents the problem of software reengineering based on object – oriented principles and the use of web services. The theoretical part describes the concept of object-oriented programming which provides such that application architecture that it can be later expanded, maintained, or modified without changing the core of the application. Prototype of application, built on the principles of object - oriented programming, was reengineered in terms of switching from one platform to another.

I. INTRODUCTION

Software systems need to be reengineered for several different reasons: business process changes, changes in requirements and policies, modernization of technological infrastructure, change of legislation, as well as because of difficulties during maintenance in large software systems.

Aim of this paper is to present a case study of reengineering that use object-oriented principles to transform desktop application into web version. The example shows possibility to have the application core functions implemented with class libraries and the front-end platform shift (from desktop to web) was therefore easy to implement.

The rest of the paper is organized with sections as follows: Theoretical background in software reengineering, object-oriented programming and SOLID principles, Literature review, Research methodology, Research results – case study and conclusion.

II. THEORETICAL BACKGROUND

A. Software Reengineering

Reengineering is process in software development that includes improvements of existing solutions for better maintenance support, performance improvements, architectural change, technology platforms adaptations etc. During the software reengineering process, system is being reconstructed into a new form.

Reengineering includes sub-processes (Figure 1): reverse engineering, code reconstruction, data reconstruction, forward engineering, inventory analysis, document reconstruction. [1]

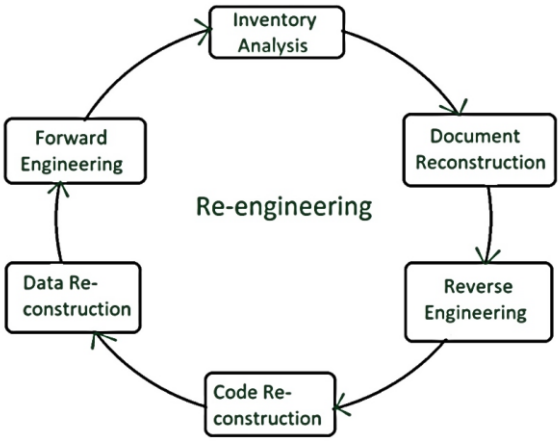

Figure 1. Software reengineering steps [1]

One of software systems types that, after a long period of use, undergo reengineering, are legacy systems. They could be defined as usually large software systems with great importance [1, 2] Legacy system is an old software program in use, still satisfying user needs, despite the fact that there are newer technologies or more efficient methods for performing the task. Their usage is closely related to procedures and terminology that were used for many years and reengineering of such systems is a complex task, not only because of the large number of software functions, but also because of the domain knowledge needed to be understood. Therefore, new developers sometimes have difficulties to understand the system. The large organizations continue using such systems because there are crucial for the business. [3]

The goal must be set in software reengineering and understanding of existing program is crucial. It is beneficial to implement the concept of software reuse within the reengineering, where having reusable modules (such as class libraries) could

speed-up the process. With high quality components, software reuse can simplify the code and make it reliable. [4]

Reengineering can include use of documentation for reconstructing and reorganizing the existing system, as well as reverse engineering to acquire models and sources suitable for reconstruction. The overall functionality of the system does not need to be changed, usually only the system architecture is changed. Reengineering means renewal, it is the process of transforming the old system into a new system, but it also could include addition of new functionalities. Special concern, during reengineering is to create such a software that will have components ready for future reuse and better maintenance. Regarding the reengineering model: each reengineering starts, if possible, with the source code of the existing application and the target goals of what is desired achieve. At this stage, the existing system can be translated into a new one by switching from one programming language to another, from one operating system to another, or from one platform to another. [5]

B. Object-oriented programing

OOP is programming model based on object concepts which are used in real life. Objects contain data in form of attributes and code in form of methods. Computer program constructed in this way have classes. Objects are instances of classes. [6] There are four basic principles supported object – oriented programming languages:

- Abstraction - using an abstract class / interface shows the purpose of the class, not its implementation. An abstract class has one or more abstract class members. An abstract class member is one that is not implemented, it has only a declaration, not a body.
- Encapsulation is a concept that defines class protection. The information in the class is protected from direct access and the only way to change it is through defining methods. Encapsulation is achieved by dividing class members into public, private and protected. [7]
- Inheritance – one defined class (super-class) can transfer its data and functionality to a new one (sub-class). It inherits all public/protected members of the class and introduces its new specific fields and methods. In this way, a hierarchical hereditary line is created. [7, 8, 9, 10]
- Polymorphism is a property that the same method works differently, depending on the context, i.e. type of objects. It is made possible with the ability of a subclass to redefine the inherited method. [7, 9, 10, 11]

C. SOLID principles

SOLID is acronym for five principles of design of object-oriented code. It doesn't teach developers how to create programs, but it helps them to create better written code. [19]

- *Single Responsibility Principle* - one class or module should be responsible for one part of the functionality provided by the software, and that responsibility should be fully supported in the class. [20]
- *Open/closed principles* - Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. This means that modules should be written in that way that they can be expanded without the need to modify them - to be able to change what the module does, without changing the source code. [21]
- *The Liskov Substitution Principle* - extends the Open/Close principle. The focus is on behavior of super classes and their derived types. Objects in the program should be interchangeable with examples of their subtypes without changing the correctness of that program. [12]
- *Interface segregation principle* - The goal of the principle is to reduce the side effects and frequency of change requests by splitting the software into several independent parts. This is only feasible if an interface is defined that corresponds to a specific client or task. [13]
- *Dependency Inversion Principle* - abstraction does not depend on details, but details depend on abstraction. In other words, the same level of abstraction that is at a given level should be used. Interfaces may depend on other interfaces and no concrete classes should be added in the signature method of an interface. [14]

III. LITERATURE REVIEW

When it comes to software reengineering, there are many examples of applying its principles in practice. Almost every company that is successful in the field of programming has at least one legacy system that needs reengineering.

"When software is developed over a long period of time, it is often very difficult and expensive to follow all standards all the time. Therefore, reengineering is performed, which improves the functionality and efficiency as well as the sustainability of the code. Existing applications can be adapted to improve their own functionality, user interface, while taking full advantage of the current

technology. This way results in applications becoming more reliable, efficient and easier to use, and all initial client investments remain fully preserved." [15]

A. Transforming desktop applications to web applications

Internet is becoming a platform on which people work, communicate, share and collaborate, with using mobile applications, web applications and even desktop applications. The largest Internet browsers such as Google and Yahoo support the gradual replacement of desktop applications with a new brand of web-based applications. Although this is a justified approach and there are some successful applications that have changed the platform (email, bookmark…) using today's web technologies to re-create sophisticated desktop applications is very difficult. Newly created web-based office applications come with significant limitations. They may be fun and convenient, but they are far from efficient, flexible, and productive as desktop applications with well-established databases. On the other hand, web applications offer many advantages, enable data sharing, wide access, low risk of data loss and most importantly cooperation. [16] There are many other examples of applying the principle of software reengineering in practice. Almost every company that is successful in the field of programming owns at least one legacy system that needs reengineering.

Vasyl Soloshchuk, SEO developer of INSART - Fintech Engineering described on its linkedIn profile successfully completed software reengineering projects. Online marketing platform has a history of more than ten years become very complicated to update and new requests indicated that it should be executed complete reengineering. With the help of the latest technologies, a new system architecture has been created that improves performance. It also avoids all errors found in legacy system and new functionalities are added to it. The result was a system which has become scalable and very easy to update. [17]

NASA has a whole complex of legacy systems that are becoming very expensive for maintenance and reengineering is one of the processes that can modernize these systems. In order to maintain the systems later, at the lowest cost, modern principles of software engineering are used. A branch dealing with software technologies in the NASA / Johnson Space Center has been involved in development and testing for reengineering methods and tools for several years within several large projects. One of them is ROSE (Reusable Object Software Environment) project that represents reengineering FORTRAN in object - oriented C ++. [18] Reengineering is a combination of reverse engineering, followed by advanced engineering into a new modernized software system (Figure 2). Three levels of reengineering are presented: translation, redesign and complete reengineering. [18] On the ROSE project, reverse reengineering was used for recovery of the FORTRAN design. Several CASE tools were used, such as data and structure analysis tools, complex metric tools, restructuring tools... The forward engineering part of this process used Object Modeling Techniques, Object Oriented Analysis Development Methods and design. [18]

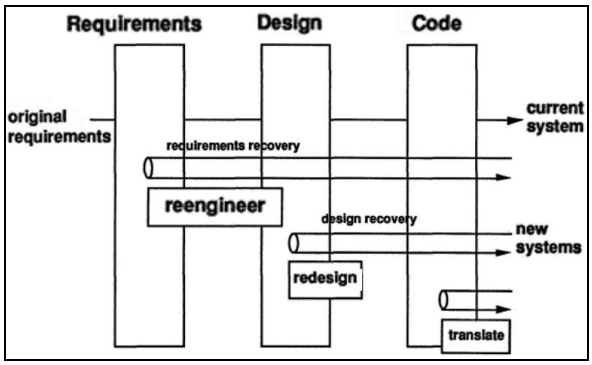

Figure 2. Software reengineering through process on ROSE project [18]

IV. RESEARCH METHODOLOGY

The research presented in this paper was conducted in year 2019. The subject of research is to theoretically and empirically explore the possibility of applying the basic principles of object-oriented programming (encapsulation, inheritance, and polymorphism) and SOLID principles when creating a new software version, i.e. software reengineering. Research hypothesis is formulated in the sense of proving the possibility of application of OOP and SOLID principles in aim to enable transformation of desktop to web application with particular concerns on structural quality of programming code.

Two types of research were conducted in this paper - theoretical and empirical. Theoretical research was supposed to answer the question of the principles of object - oriented programming used in implementation of the software to make it suitable for later eventual change of platform while preserving the functional core of the application. Empirical research is based on prototyping a multilayer desktop application and transforming into web application. Within this development, a new version of the software is in focus, with transition

from one platform to another (desktop application to web application) while preserving the functional core of the application. The essence of using OOP principles is to use the same classes from middle tiers, to build a web version of the application.

A. Description of the business context

The application business domain chosen to illustrate the concepts from the title of this paper is recording data about university teaching staff. Data about new teaching staff are entered into a database, and the type of teaching staff is checked. Depending on the type some data are enabled or prohibited. In this case, there are two types teaching staff: Teacher (professor) and Associate. If an associate is concerned, application enables only general data entry, while in professor case additional data are required to be entered.

B. Aim and description of the used technologies

The aim of the practical part of this paper is to show the object - oriented principles in creating software for later reengineering in order to move from one platform to another with the change of technologies for creating a user interface (desktop and web).

Tools / Development environments used for implementation examples are: MS SQL Management Studio and MS SQL Server, Visual Studio 2019, as well as Sybase Power Designer. The languages are SQL for database generation, C# for application creation and UML for model creation.

V. RESEARCH RESULTS – CASE STUDY

The application consists of four parts: database, web service, user interface and the program code which is the core of the application. All of these parts are physically separated, but through references they work together. As the desktop application is created as an object-oriented n-tier application and each part is independent, reengineering was quite simple. It was only front-end, i.e. new user interface that was created, but it was attached to existing components. These two examples (desktop and web applications) use the same crucial parts. The component diagram Figure 3 presents the component diagram, which shows the whole solution.

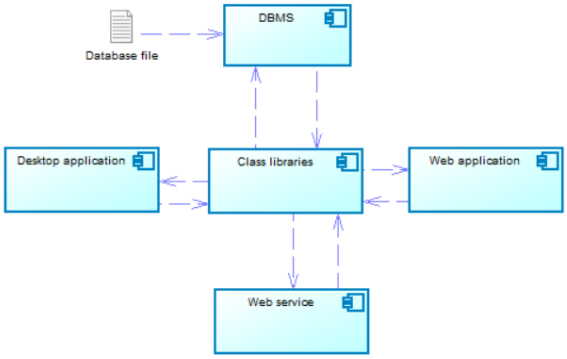

Figure 3. Component diagram

In Deployment diagram, since there are two versions of the application, desktop and web applications that are based on the same core, there is no significant difference in the layout diagram. The only difference is that with a desktop application (Figure 4), the software runs directly on the client computer, and the web application (Figure 4) is running on web server and presented with web browser.

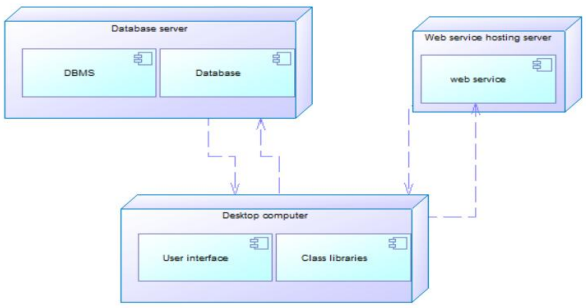

Figure 4. Deployment model for desktop application

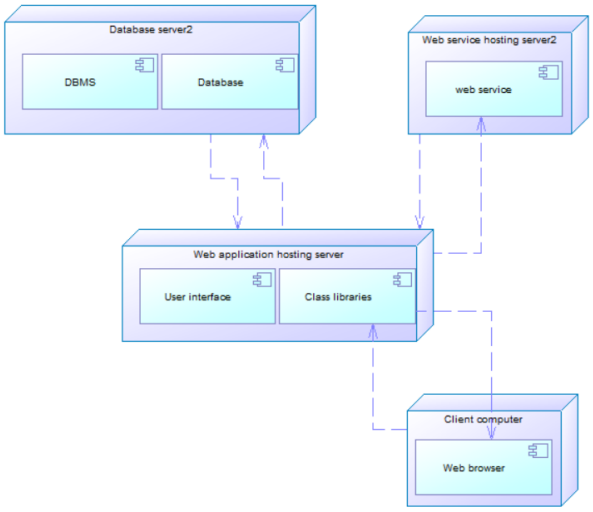

Figure 5. Deployment model for web application

For the purpose of this work, two prototypes of the application were made, one is a desktop version (Figure 6) and another web version (Figure 7) of the same application. Both applications implement the same functions. They allow the entry of teaching staff. Anyone can enter first and last name, work position, and depending on that, other fields are opened. If the position is "Teacher", it is possible to choose the title and scientific field. If he is an associate, there is no possibility of further data entry. The entered data can be stored in the database or the action can be canceled.

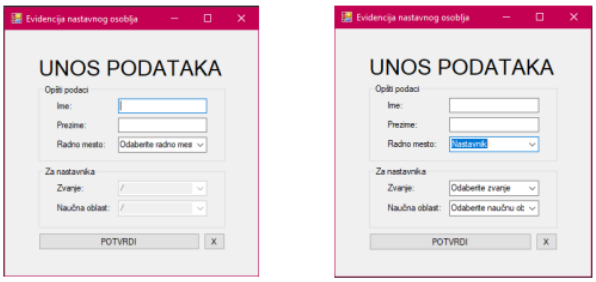

Figure 6. Desktop application user interface

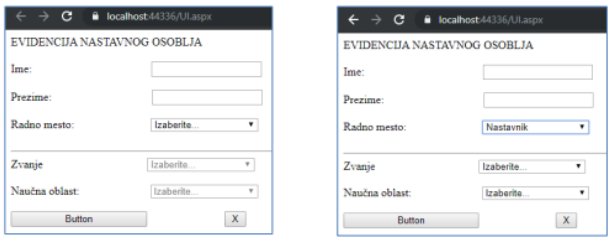

Figure 7. Web aplication user interface

VI. CONCLUSION

The most common reengineering is the reengineering of outdated software that has a good function but due to their technologies are no longer subject to change and are expensive to maintain. Reengineering helps to retain their core functions and support business processes.

Aim of this research is to explore the role of object-oriented programming principles in efficiency of reengineering. During the theoretical research and related work analysis, it has been proved that object-oriented programming and use of SOLID principles, as well as n-tier architecture helps in more efficient software reengineering. Therefore, hypothesis was proved. Another proof for hypothesis has been obtained with the implemented practical example – case study.

During the practical development of the application prototype, the principles described theoretically were included and it was concluded that such an application is possible, which, once again, proves the hypothesis. The application is organized through layers, to make the transition from platform to platform easier. It is built so that it has a user interface independent and all methods are inherited from other classes. Classes are divided by function, according to what they do and through the references included in the system. The database is also independent and can be easily changed at any time. Such an organization has proven successful in transitioning with desktop to web version of a software.

REFERENCES

[1]https://www.geeksforgeeks.org/software-engineering-re-engineering/
[2] L. Tahvildari, Quality-Driven Object-Oriented Re-engineering Framework, 2003
[3] P. Tripathy, K. Naik, Software Evolution and Maintenance, Wiley 2015
[4] R. J. Leach, Software reuse: Methods, Models and Costs, 2011
[5] H. Singh, Software Reengineering: New Approach to Software Development, International Journal Of Research In Education Methodology, 2012.
[6] https://www.geeksforgeeks.org/differences-between-procedural-and-objectoriented-programming/
[7] https://radmiladrakulic.wordpress.com/2017/09/27/osnovni-principi-objektnoorijentisanog-programiranja/
[8] https://www.upwork.com/hiring/development/object-oriented-programming/
[9] https://cubes.edu.rs/sr/30/obuke-i-kursevi/sta-je-objektno-orijentisanoprogramiranje
[10] https://medium.com/@cancerian0684/what-are-four-basic-principles-of-objectoriented-programming-645af8b43727
[11] https://medium.com/from-the-scratch/oop-everything-you-need-to-knowabout-object-oriented-programming-aee3c18e281b
[12] https://codeburst.io/the-liskov-substitution-principle-5ba387055a2a
[13] https://stackify.com/interface-segregation-principle/
[14] https://williamdurand.fr/2013/07/30/from-stupid-to-solid-code/#prematureoptimization
[15] P. Kaur Chahal, A.Singh, Software Reuse and Reengineering: With A Case Study, International Journal of Soft Computing and Engineering (IJSCE), 2014.
[16] H. Shen; Z. Yang; C. Sun, Collaborative Web Computing: From Desktops to Webtops, IEEE Distributed Systems Online, 2007
[17] V. Soloshchuk, Three Examples of Successful Software Reengineering Implementation, linkedIn, 2016 https://www.linkedin.com/pulse/software-reengineering-successfulimplementation-vasiliy-soloshchuk-1/
[18] C. Pitman, D. Braley, E. Fridge, A. Plumb, M. Izygon, B. Mears, Reengineering Legacy Software to Object – Oriented Systems, NASA
[19] G. Kumar Arora, SOLID principles, Syncfusion, 2016.
[20] J. Ocampo, J. Meridth, Pablo's SOLID
[21] Software Development, LosTechies.com, 2009
R.C. Martin, Design Principles and Design Patterns, 2000