# The presence of design patterns in software - A research of the Spring Framework

Niklas Pettersson
Department of Computer and Information Science
Linköping University
Linköping, Sweden
nikpe353@student.liu.se

Nils Petersson
Department of Computer and Information Science
Linköping University
Linköping, Sweden
nilpe995@student.liu.se

***Abstract*—Importation, usage of extensions and reusing code is a big part of software development. While focusing on the application as a whole, a developer can feel safe and secure by using frameworks that provides solutions for them. One framework like that is the Spring framework. Being made up of combinations of different structures and skeletons for reusage of code, the Spring framework is one of the most popular framework for the java language. What often is missed is that the frameworks are built on the foundation of different design patterns. Patterns that developers might take for granted but provides solutions to problems unaware of.**

***Keywords* -** Design patterns, Spring MVC, object-oriented programming

## I. Introduction

While implementing a web application, software developers can rely on the help of independent frameworks for structuring and implementation techniques. One can also find that by using these well documented extensions and certain parts of frameworks, a developer can make sure their project is of high quality and also save a lot of time in the process [1]. What might go unnoticed in the developing phase is the presence and usage of well known design patterns.

### A. Background

When using an extensive framework a lot of things comes for free, however this might in some cases make it harder for the developer to understand certain things such as the underlying architecture of the code.

In order to understand these underlying causes and architecture of code, one should investigate the structure of the already written code and how solutions were implemented. Often, but not always, one might find that design patterns is how these structures and solutions came to be.

### B. Research description

As earlier stated, when using a framework things might happen in the background without the knowledge of the developer. In this paper we will investigate how design patterns are being used in new software technologies, limited to the perspective of using the Spring MVC framework. Therefore, this paper will focus on the following research questions.

*How are are design patterns being used in the background when developing applications with the help of Spring MVC?*
*Are there other design patterns embedded in the Spring framework?*

## II. Theory

In order to fully understand this paper some things need clarification. Therefore, in this chapter, important terms will be further explained. Certain names and words, such as abbreviations, might also be considered as important information, but are instead described in the text.

### A. Design patterns

Amongst software developers the phrase *Design pattern* is widely recognized. The term refers to the reusage of a solution to an implementation problem regarding software design. The design itself is not a complete structure in any way, but merely functions as a template or description of how the code could be structured. Design patterns could therefore help solve common problems in many different situations [2]. The patterns are commonly divided into three categories; Creational pattern, Behavioral pattern and Structural pattern.

*1) Creational:* Creational design patterns are patterns that handles the mechanisms of creating object. It does this, in regards to either basic forms or with more complex problems of creation, by simply controlling the creation in a suitable fashion [2]. One example of such a design pattern is the Singleton pattern.

*2) Structural:* Structural design patterns are patterns that aims to simplify the design of software by handling the relationship between objects (entities) [2]. One example of a structural design pattern is the Proxy pattern.

*3) Behavioral:* Behavioral design patterns are patterns that handles the communication between objects. Common communication ways are identified and then realized into patterns with the intent to increase the flexibility regarding the communication [2]. One example of such a pattern is the Template method pattern.

### B. Spring

Spring is an easy to use open source application framework and IoC[1] container [3]. It is continuously updated and was first released by Ron Johnson in October, 2002. The spring framework is widely used in the software industry and has been used to create many big applications such as LinkedIn (which have parts of their service built on Spring MVC [4]).

[1]IoC stands for Inverison of Control and is a design technique in software development that reverses the flow of control in a program. The Hollywood principle ”Don’t call us, we’ll call you” is commonly used to describe IoC.

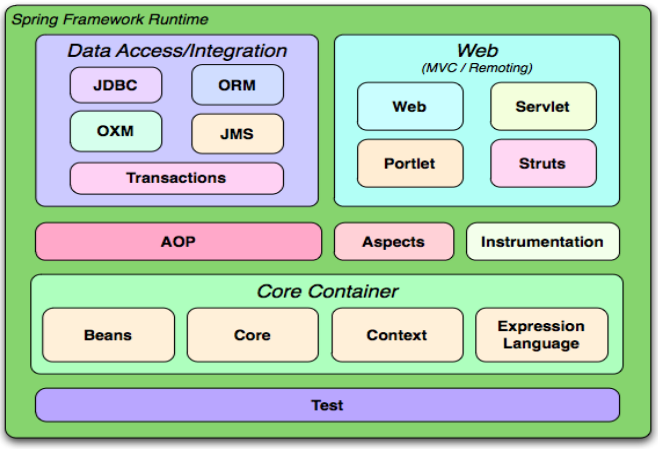


Fig. 1. The Spring projects [5]

Spring consists of many different features that can be used to build all types of applications, and contains extensions suitable for web development. The framework is divided into about 20 modules, as can be seen in Fig. 1 above. For example, the core container includes the basic parts of the framework, such as beans and the core module. The core module itself includes the IoC and dependency injection features. Spring also features a test module that supports testing with, among many others, JUnit[2] or TestNG[3] to easier test the spring components [7].

### C. Maven

Apache Maven is a software tool for project management. By using the project object model (pom) one can manage the build, documentation and reporting of the project in a centralized manner. The pom file is written in XML-format and is describing the dependencies, software build, directories and external components and modules. The pom file could also contain information regarding build order and plug-ins in the project [8].

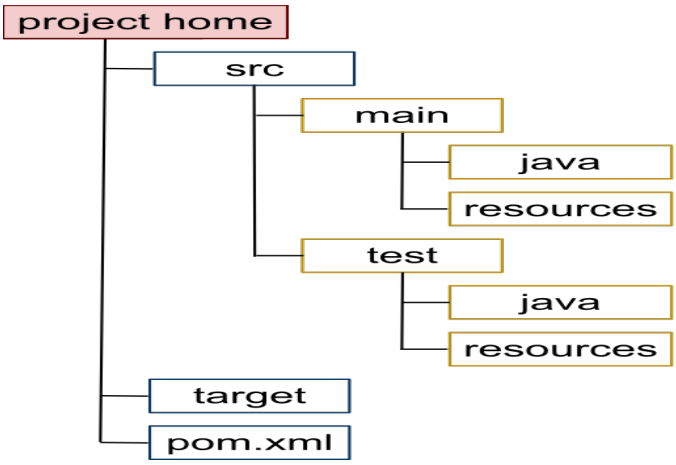


Fig. 2. The autogenerated filestructure provided by Maven.[9]

[2]JUnit is a simple framework for testing. It is built on the xUnit architecture used for usnit testing frameworks.[6]

[3]TestNG is a framework for testing inspired by JUnit but includes some additional functionality such as Annotation etc.

### D. Tomcat

The Apache Tomcat software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. Apache Tomcat was first released in 1999 and has since been developed and maintained by an open community of developers as until today. New functionality is therefore added regularly [10].

### E. MVC

Model View Controller (MVC) is a software architectural pattern that separates data (model), presentation (view) and logic (controller) in a user interface. It is widely used in software development for web applications and Graphical User Interfaces (GUIs) [11].

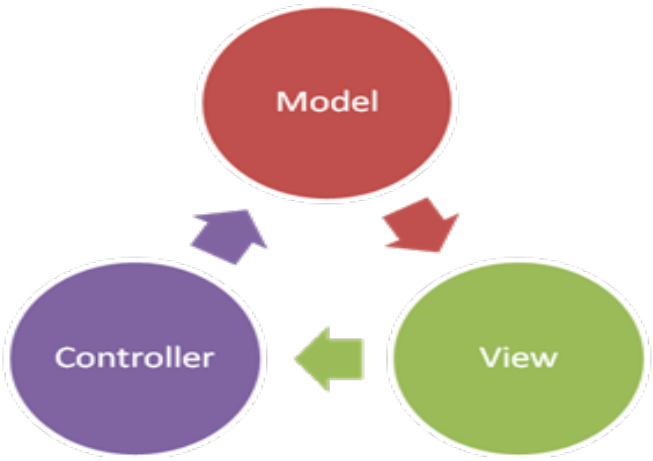


Fig. 3. The three MVC components.[12]

## III. Implementing a web application - Phonebook

This section will give an explanation on how we went on with implementing a web application with the purpose of examining the different design patterns being used when using the spring framework.

### A. Phonebook

Phonebook is a web application that lets users store information regarding their contacts such as name, email, number and additional information. The application was written in the Java language and was implemented with the help of the Spring framework. The project structure and dependencies were formatted with the help of Maven. In order to utilize Maven, the IDE (integrated development environment) Eclipse was used [4].

With the help of Spring MVC, the application was based on the architectural design MVC. That meant implementing models, for example *Contact.java*, controllers, for example *ContactController.java*, and views, for example *menu.jsp*, so that Spring could handle incoming request and dispatch them correctly (see Fig.4 below). By doing so, the development was mainly focused on implementing new features and adding functionality.

[4]Eclipse is a development tool for programming that is widely used among private users and in the software industry.

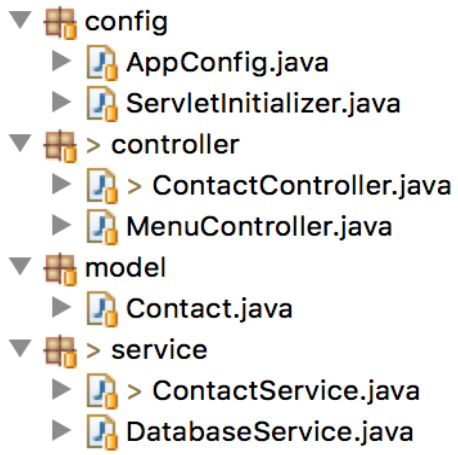


Fig. 4. Project structure for Phonebook

When implementing new features, such as integration with a database to store contacts in, the only detail that needed to be added was the dependency of a package in the pom.xml file (along with the imports in the .java file itself). This resulted in much easier implementation since both the dependency and gathering of files (usually done manually when programming) was done by maven in the background.

In order to run our web application, Tomcat was used. More specifically tomcat's servlet container called *catalina*. When building the project using maven a .war file was generated and placed within the tomcat's webapps folder in order to run.

### B. Spring MVC

Spring MVC uses the same style of implementing an application as MVC but with little variations. The controllers itself are simple classes, POJOs (Plain Old Java Objects), which simplifies the implementation of each controller in many aspects. Spring MVC also uses a *front controller*, which means that a controller is placed in front of all the other controllers, as a centralized entry point to the system, handling the incoming request [13].

## IV. Pattern analysis of Spring Framework

In this part of the paper, examination of our own, as well as external sources, view on the design patterns being used within the spring framework is conducted. This will be done by linking general behaviors from a design pattern to the actual functionality of a Spring object and thus showing the existence of a certain pattern.

### A. Singleton

The singleton pattern is when creation of a class only occurs once. This is usually handled by a separate class in order to make the control of creation and ways of accessing the created class easier. This pattern makes it so that accessing a class can be done without the need to instantiate an object. As earlier stated in the section about design patterns, singleton is considered as a creational pattern [14].

Objects that are only created once in Spring MVC is the bean objects that are defined in the configuration file for Spring (default set as singleton) [15]. A bean object is, as defined by Spring, *"When you create a bean definition what you are actually creating is a recipe for creating actual instances of the class defined by that bean definition. The idea that a bean definition is a recipe is important, because it means that, just like a class, you can potentially have many object instances created from a single recipe"* [16][17].

Listing 1. Example of an instance of a Bean-object.

```
@Bean
public ViewResolver jspViewResolver() {
    InternalResourceViewResolver resolver =
    new InternalResourceViewResolver();
    resolver.setViewClass(JstlView.class);
    resolver.setPrefix("/view/");
    resolver.setSuffix(".jsp");
    return resolver;
}
```

As seen above, in Listing 1, the ViewResolver is created as a bean object. This was done in the phonebook-application and it created a singleton object for the views (the presentation-layer to the user). What this meant, for the funcitonality of the application, was that spring had information on how incoming request should be handled. More specifically, how information should be displayed to the user (through a .jsp-file[5]).

### B. Template method

The Template method pattern is a behavioral pattern that consists of an abstract class that define how to execute its methods. The subclasses can override these implementation if needed or keep the superclass version of it [19].

In spring this can be used, for an example, to minimize boilerplate code for when closing database connections by using the jdbcTemplate package [20].

Listing 2. Example of an JDBCTemplate usage [21].

```
public Customer findByCustomerId(int custId){
    String sql = "SELECT * FROM
        CUSTOMER WHERE CUST_ID = ?";

    Customer c = (Customer)getJdbcTemplate()
        .queryForObject(sql, new Object[]
        { custId }, new CustomerRowMapper());

    return c;
}
```

As showcased in Listing 2 above, the usage of jdbcTemplate code is very useful. Three lines of code creates the entire connection and closing statement by just using already existing functions provided.

Listing 3. Code for database handling (Phonebook).

```
Connection c = null;
Class.forName("org.sqlite.JDBC");
c = DriverManager.getConnection(
    "jdbc:sqlite:" + dbName);

String sql = "SELECT ID, NAME,
    MAIL, NUMBER, INFO FROM CONTACTS";
Statement stmt  = c.createStatement();
ResultSet rs = stmt.executeQuery(sql);

while (rs.next()) {
    // set the variables in contact
}
c.close();
```

[5]JSP stands for JavaServer Pages and is a technology that lets software developers create dynamic web content based on, for example, HTML and XML [18].

Comparing the jdbcTemplate to Listing 3 above, where a minimum of nine lines of code is used (excluding the lines of code in the while-loop), showcases the compactness and convenient of template usages.

### C. Proxy

When using the proxy pattern, a structural pattern, we have another class to act as an interface for an object. A proxy can be used as a simple wrapper just forwarding information to the real object or in other cases also modify this information or decide to not forward certain bits. [22].

Spring AOP (Aspect oriented programming)[6] operates by creating a proxy around the service beans and because of this, Spring AOP only works together with spring beans [23][24].

### D. Factory

When using the Factory pattern we create objects without revealing the logic for creating it to the client and uses a common interface to refer to it [25]. The Factory pattern is considered to be a creational pattern.

Spring uses The Factory pattern when creating bean objects using the BeanFactory[7]. The BeanFactory is considered a main component of the core container, which uses the IoC technique in order to separate dependency specifications and configuration [26].

### E. Dependency injection

This is a more of a technique rather than a design pattern, but is still classified as a design pattern, were objects supply other object with their dependencies. The dependency is referred to as a service, while the injection, or the passing, of an dependency to a dependent object is called a client. This means that the client is not able to build or find a service, but is being passed the service to it.

With maven, the dependencies is declared in the pom.xml file (see Listing 4. below) and is loaded into the project when the project is being built [27].

Listing 4. One dependency in the pom.xml file (phonebook application).

```
<!-- Spring MVC -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
</dependency>
```

Dependency injection could also refer to the usage of getters and setters in object-oriented programming (the mutator method). The goal of this concept is to establish encapsulation by making variables and fields in classes private. In order to change or access these fields other objects uses the setters (modify the value of the field) and getters (retrieving the value of the field) [27].

## V. Discussion

While discussing the result and answering research questions is of high priority in this paper, validating and reviewing the method used thoroughly functions as a corner stone in the level of correctness of the paper. The same could be stated regarding the sources used. Because of this, the following chapter will discuss not only the gathered result, but also the method and the sources.

[6]Aspect oriented programming, a programming paradigm that is allowing the separation of cross-cutting concerns in order to increase modularity[**AOP**].

[7]BeanFactory contains multiple definitions of beans and instantiates a bean when requested.

### A. Method

Firstly, the method used in this paper might seem unconventional and unrealistic in order to reach a valid answer; answering the previously stated research questions. With the produced result in mind though, the method has to be considered as successful since several cases of design patterns has been showcased.

Secondly, what should be noticed is the size of the implemented application (Phonebook). Given more time and experience with the Spring framework the application could be extended and further developed in order to fully utilize the frameworks extensions.

Finally, the research commited in finding sources, claiming usage of design patterns in the Spring framework, is of great value to this paper. Take the Factory pattern for example. The fact that the pattern is used in Spring would have gone unnoticed if not for the findings of the source claiming it.

### B. Sources

There is an wide range of different sources presented in this paper, varying from research papers to online tutorials. Research papers that are connected with a science department of an university have to be considered as a very credible source due to publish policy and demands on quality for universities. These are the sources we strive to reference the most in this paper.

However, there are still some parts in the paper that use tutorials or guides as sources. For example when describing a design pattern or certain established technologies. This is because of the lack of research papers focusing purely on the description of a certain design pattern or technology. In general, descriptions could be referenced to using any source as long as it is a credible website, or published article, being the source. Taking a website such as *www.spring.io*, as an example, one must act as if the facts that are presented there is credible considering that it is the official website of the Spring framework.

Another point that may be good to notice is the usage of many newly published sources (just a few years old). These sources might be less trustworthy because the small time frame in which they could've been evaluated and reviewed.

### C. Result

With the acquired result as a guideline, it seems like a obvious thing to establish that there a lot of design patterns being used in the Spring framework. Some observations of design patterns might be more obvious than others. For example, the MVC pattern might be considered as redundant information since the implemented application is using the Spring MVC extension. Even when discarding this as obvious, it is still in fact a widely used design pattern being present in the framework. Common sense might also exclude some of the results provided that is just a small example, such as the dependency injection used with the Maven framework. What is to be noted in this case is that the Spring framework heavily relies on the existence of injection of dependencies in several cases. One case like that could be when creating a class in Spring, the dependency is often provided from another object or class, usually a .xml file, when the class is requested (instantiated). But since that example is not present in the implementation of Phonebook, the maven example was used instead as an illustration.

Considering the fact that the Singleton, Template method, Proxy, Factory and Dependency injection pattern is brought up in the result, the questions stated in the beginning of this paper is both answered and researched to a certain extent. Similar to when G.Hohpe discusses the existence of design patterns in SoA patterns, one could argue

that design patterns is heavily present in a framework such as Spring [28]. By not only seeing the result of this paper as a foundation for that statement, but also including the fact that the Spring online documentation brings it up several times, the observations made becomes utterly clear: Design patterns are present in the Spring framework. Both intentionally and unintentionally.

### *D. Importance of design patterns*

Design patterns serve many different purposes in the software industry ranging from a common vocabulary to simplifying testing of systems[28]. By ”capturing” the structure of code in words, even very complex structures can be reused by many different developers. By improving the communication and understanding between developers, focus can be put elsewhere, thus improving the efficiency in the development process. Another positive view of using design pattern is that code that follows a specific pattern often becomes more flexible.

## VI. Conclusions

Firstly, when it comes to design patters in general, developers might not realize that their implementation is using a solution that is very close to what a well known design pattern describes. This could be the effect of the existence of so many different design patterns, but also the lack of knowledge in regards to the extension and frameworks used.

Secondly, the use and existence of design patterns might not always improve the code, and the project as a whole. When used to much, and especially when implementing several different patterns, design patterns might complicate both the process of testing and extensibility [29]. Although this often comes from an incorrect use of a pattern, it is also an important fact to be noted. By acquiring knowledge and better understanding of design patterns, one could avoid such problems.

Finally, after implementing the phonebook application, and after investigating the usage of design patterns in spring, our knowledge about the Spring framework has vastly improved. This way of reviewing and understanding code and frameworks can be applied to almost every aspect in programming and thus help programmers to further develop their talents and knowledge in the field of software.

## References


[1] Dandan Zhang, Zhiqiang Wei, and Yongquan Yang. *Research on Lightweight MVC Framework Based on Spring MVC and Mybatis*. 2013. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6805007 (visited on 05/10/2017).

[2] Erich Gamma et al. *Design Patterns*. 1995. URL: http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf (visited on 05/10/2017).

[3] Pivotal Software. *Spring framework*. 2017. URL: https://projects.spring.io/spring-framework/ (visited on 05/10/2017).

[4] Baq Haidri. *Check out how LinkedIn uses JRuby on its Front-end*. 2010. URL: https://www.youtube.com/watch?v=qZcmF3yonjs (visited on 05/23/2017).

[5] Pivotal Software. *Spring framework - Modules*. 2017. URL: http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html (visited on 05/23/2017).

[6] JUnit. *JUnit - About*. 2017. URL: http://junit.org/junit4/ (visited on 05/10/2017).

[7] Darryl K. Taft. *Spring Creator Rod Johnson Leaves VMware, Makes Mark on Java*. URL: http://www.eweek.com/development/spring-creator-rod-johnson-leaves-vmware-makes-mark-on-java (visited on 05/20/2017).

[8] Agnes F. Vandome Frederic P. Miller and John McBrewster. *Apache Maven*. 2005. ISBN: 6130652194 9786130652197.

[9] Wikipedia. *Structure of a Maven project*. 2010. URL: https://en.wikipedia.org/wiki/Apache_Maven#/media/File:Maven_CoC.svg (visited on 05/10/2017).

[10] Apache Software Foundation. *Tomcat*. 2017. URL: http://tomcat.apache.org/ (visited on 05/10/2017).

[11] Glenn E. Krasner and Stephen T. Pope. *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*. 1988. URL: http://heaveneverywhere.com/stp/PostScript/mvc.pdf (visited on 05/19/2017).

[12] Mahmoud Mohamed. *What is the true/original definition of MVC?* 2016. URL: https://qph.ec.quoracdn.net/main-qimg-5fe8c013edf4456a85967713963ac590 (visited on 05/10/2017).

[13] Prof. M.C. Govil Praveen Gupta. *Spring Web MVC Framework for rapid open source J2EE application development: a case study*. 2010. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.168.2988&rep=rep1&type=pdf (visited on 05/20/2017).

[14] Tutorialspoint. *Design Pattern - Singleton Pattern*. 2017. URL: https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm (visited on 05/10/2017).

[15] MKyong. *Spring Bean Scopes*. 2012. URL: https://www.mkyong.com/spring/spring-bean-scopes-examples/ (visited on 05/10/2017).

[16] Spring. *Bean Scopes*. URL: http://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch04s04.html (visited on 05/19/2017).

[17] Ahmet Arif Aydin. *Spring Framework and Depency injection*. URL: https://www.cs.colorado.edu/~kena/classes/5448/f12/presentation-materials/aydin.pdf (visited on 05/19/2017).

[18] Oracle. *JavaServer Pages Technology*. URL: http://www.oracle.com/technetwork/java/javaee/jsp/index.html (visited on 05/23/2017).

[19] Tutorialspoint. *Design Pattern - Template Pattern*. 2017. URL: www.tutorialspoint.com/design_pattern/template_pattern.htm (visited on 05/10/2017).

[20] Pivotal Software. *JDBCTemplate*. 2017. URL: http://docs.spring.io/spring/docs/2.5.x/reference/jdbc.html#jdbc-JdbcTemplate.

[21] Mkyong. *Spring JdbcTemplate Querying examples*. URL: https://www.mkyong.com/spring/spring-jdbctemplate-querying-examples/ (visited on 05/21/2017).

[22] Tutorialspoint. *Design Pattern - Proxy Pattern*. 2017. URL: www.tutorialspoint.com/design_pattern/proxy_pattern.htm (visited on 05/10/2017).

[23] Pivotal Software. *Understanding AOP proxies*. 2017. URL: http://docs.spring.io/spring/docs/2.5.x/reference/aop.html#aop-understanding-aop-proxies (visited on 05/10/2017).

[24] Prakash Jayanth Kulkarni Kotrappa Sirbi. *Stronger Enforcement of Security Using AOP  Spring AOP*. 2010. URL: https://arxiv.org/ftp/arxiv/papers/1006/1006.4550.pdf (visited on 05/21/2017).

[25] Tutorialspoint. *Design Pattern - Factory Pattern*. 2017. URL: www.tutorialspoint.com/design_pattern/factory_pattern.htm (visited on 05/10/2017).

[26] Weihua Hu Xiaoliang Xu. *Research on J2EE Teaching Based on Mainstream Open Source Frameworks*. 2010. URL: https://pdfs.semanticscholar.org/d681/e9cc1d8dd49f2f92312bff1db22cc7aa1bed.pdf (visited on 05/20/2017).

[27] David Janzen Ekaterina Razina. *Effects of dependency injection on maintainability*. 2007. URL: http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1035&context=csse_fac (visited on 05/19/2017).

[28] Gregory Hohpe. *SOA Patterns - New Insights or Recycled Knowledge?* 2007. URL: http://www.enterpriseintegrationpatterns.com/docs/SoaPatterns.pdf (visited on 05/10/2017).

[29] Prechelt Lutz et al. *A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions*. 2001. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=988711&tag=1 (visited on 05/23/2017).