REPUBLIC OF TURKEY

BAHÇEŞEHİR UNIVERSITY

THE IMPORTANCE OF DEFINING CODE STANDARDIZATION AND CREATING BASE FRAMEWORK FOR COMPANIES

Master Thesis

YUNUS EMRE KIRKANAHTAR

İSTANBUL, 2018

REPUBLIC OF TURKEY

BAHÇEŞEHİR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPUTER ENGINEERING

THE IMPORTANCE OF DEFINING CODE STANDARDIZATION AND CREATING BASE FRAMEWORK FOR COMPANIES

Master Thesis

YUNUS EMRE KIRKANAHTAR

Thesis Supervisor: ASSIST. PROF. MUSTAFA EREN YILDIRIM

İSTANBUL, 2018

REPUBLIC OF TURKEY
BAHÇEŞEHİR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

Name of thesis: The importance of defining code standardization and creating base framework for companies
Name/Last Name of Student: Yunus Emre Kırkanahtar
Date of Defense of Thesis: 28.05.2018

The thesis has been approved by the Graduate School of Natural And Applied Sciences

Assist. Prof. Yücel Batu SALMAN
Graduate School Director

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Arts.

Assist. Prof. TARKAN AYDIN
Program Coordinator

This is to certify that we have read this thesis and we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Arts.

Examining Committee Members	Signature
Thesis Supervisor Assist. Prof. Mustafa Eren YILDIRIM	----------------------------------
Member Assist. Prof. Yücel Batu SALMAN	----------------------------------
Member Assist. Prof. S. Serkan GÜLLÜOĞLU	----------------------------------

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to my wife and my family for their spiritual support and to my thesis supervisor, Assist. Prof. Mustafa Eren Yıldırım for his precious contribution during the period of this work.

İstanbul, 2018 Yunus Emre KIRKANAHTAR

ABSTRACT

THE IMPORTANCE OF DEFINING CODE STANDARDIZATION AND CREATING BASE FRAMEWORK FOR COMPANIES

Yunus Emre Kırkanahtar

Computer Engineering

Thesis Supervisor: Assist. Prof. Mustafa Eren YILDIRIM

May 2018, 92 pages

With the boost from technology, dramatic changes and developments are seen in almost any industry. Such developments also have an impact on software projects as on every unit of any company. When a software project coded according to specific needs is developed without complying with certain coding standards ignoring future changes and developments, it causes the project to fail in either short or long term while also resulting in huge financial losses. Additionally, if more than one developer works on a project and a certain standard is not followed, it is highly probable that each of these developers will develop a code of their own style. A project with codes developed in different styles makes code readability difficult, improves costs for error fixes as well as development.

One of the most appropriate solutions to minimize such problems in companies is to use frameworks developed according to certain standards. Thanks to frameworks, developers cannot go beyond certain standards and projects will have flexibility, security, and extensibility characteristics.

The first chapter of this thesis explains how an ideal software development project should be designed. Various studies conducted on this and accepted principles are reviewed.

In the second chapter, the importance of using a modular, ruled framework that includes the principles outlined in the first chapter are laid out for a company that produces various projects with a defined framework. In addition, advantages of using a ready-made framework or coding a company-specific framework are listed, as well.

In the final chapter, a sample framework study is given. The modules that should be in a standard project are identified and these modules are included in the framework study. At the end of the chapter, achievements made through a framework during development processes are analyzed.

Keywords: Software Development, Framework, Coding Standards, Clean Coding, Code Smells, SOLID Principles

ÖZET

KURUMLAR İÇİN FRAMEWORK KULLANIMININ VE KODLAMA STANDARTLARINA UYMANIN ÖNEMİ

Yunus Emre Kırkanahtar

Bilgisayar Mühendisliği

Tez Danışmanı: Dr. Öğr. Üyesi Mustafa Eren YILDIRIM

Mayıs 2018, 92 sayfa

Günümüzde teknolojinin de desteği ile her sektörde büyük değişimler ve gelişmeler yaşanmaktadır. Bu değişimler firmalardaki her birimleri olduğu gibi; yazılım projelerini de etkilemektedir. Belli ihtiyaçlara göre yazılmış bir yazılım projesi, değişim ve gelişimlere kapalı, belli kodlama standartlarına uyulmadan geliştirilmişse, kısa ya da uzun vadede projenin başarısız olmasına yol açmakta ve çok büyük maddi kayıplara sebep olmaktadır. Ek olarak bir projede birden fazla geliştirici çalışıyorsa ve belli bir standart takip edilmiyorsa, bu geliştiricilerin her birinin kendi tarzında kod geliştirmesi yüksek bir ihtimaldir. Bir proje içerisinde farklı tarzlarda geliştirilmiş kodların bulunması, kodun okunabilirliğini zorlaştırır, hata düzeltmeleri ve geliştirme maliyetlerini yükseltir.

Bu sorunları asgariye indirmek için en uygun çözümlerden biri firmalarda belli standartlara göre geliştirilmiş framework kullanılmasıdır. Framework sayesinde, geliştiriciler belli standartların dışına çıkamayacaklar ve çalışılan her proje esneklik, güvenlik ve genişleyebilirlik gibi özelliklere sahip olacaktır.

Bu tez çalışmasının ilk bölümünde, ideal bir yazılım geliştirme projesinin nasıl tasarlanması gerektiği anlatılmıştır. Bu konuda yapılan çeşitli çalışmalar ve kabul edilmiş prensipler incelenmiştir.

İkinci bölümde, framework tanımı yapılmış ve çeşitli projeler üreten bir firma için, ilk bölümde anlatılan prensipleri içeren, kurallı, modüler bir framework kullanmanın önemi anlatılmıştır. Hazır bir framework kullanmak ya da firmaya özgü bir framework yazmanın avantajları sıralanmıştır.

Son bölümde, örnek bir framework çalışmasına yer verilmiştir. Standart bir projede olması gereken modüller belirlenmiş ve framework çalışmasında bu modüllere yer verilmiştir. Bölüm sonunda ise, framework sayesinde yazılım geliştirme sürecindeki kazanımlar analiz edilmiştir.

Anahtar Kelimeler: Yazılım Geliştirme, Framework, Kodlama Standartları, Temiz Kodlama, SOLID Prensipleri, Code Smells

CONTENTS

TABLES viii
FIGURES ix
ABBREVIATIONS xi
1. INTRODUCTION 1
2. CODE STANDARDIZATION AND SOFTWARE QUALITY 4
2.1 CODE STANDARDIZATION 4
2.2 CODE SMELLS 5
2.2.1 The Bloaters 6
2.2.1 Object Oriented Abusers 8
2.2.2 Change Preventers 10
2.2.3 Dispensables 10
2.2.4 Couplers 12
2.3 CLEAN CODING 13
2.3.1 Naming Convention 13
2.3.2 Functions & Methods 16
2.3.3 Comments 17
2.3.4 Formatting 18
2.3.5 Error Handling 20
2.3.6 Unit Tests 21
2.3.7 Classes 24
2.4 SOLID PRINCIPLES 25
2.4.1 Single Responsibility Principle 26
2.4.1 Open/Closed Principle 29
2.4.2 Liskov Substitution Principle 32
2.4.3 Interface Segregation Principle 34
2.4.4 Dependency Inversion Principle 38
2.4.5 Other Principles 40

2.5 DESIGN PATTERNS 40
3. FRAMEWORK 45
3.1 WHAT IS A FRAMEWORK? 45
3.2 ADVANTAGES OF USING FRAMEWORKS 47
3.3 RELATED WORKS 48
3.4 ANTICIPATED MODULES IN CUSTOMIZED FRAMEWORKS 49
3.4.1 Authorization, Authentication and Security 49
3.4.2 Logging Transactions 53
3.4.3 FTP Transactions 53
3.4.4 Database Transactions 53
3.4.5 Web Service Util Library 54
3.4.6 Other Modules 56
4. FRAMEWORK CASE STUDY 57
4.1 DATABASE MODULE (CUSTOMFRAMEWORK.DATA) 58
4.2 BASIC AUTHORIZATION MODULE (CUSTOMFRAMEWORK.AUTHORIZATION) 64
4.3 SHARED MODULE USABLE IN ALL PROJECTS (CUSTOMFRAMEWORK.UTILS) 66
4.4 WEB API TEMPLATE (CUSTOMFRAMEWORK.WEBAPIUTILS) 67
4.5 WEB API UTIL WITH AUTHORIZATION AND AUTHENTICATION (CUSTOMFRAMEWORK.WEBAPIUTILS.AUTHORIZATION) 70
4.6 SAMPLE WEB API (CUSTOMFRAMEWORK.SAMPLEWEBAPI) 72
4.7 CODE GENERATOR 72
4.8 FTP AND E-MAIL MODULE 73
5. RESULTS 74
5.1 SAMPLE FRAMEWORK DEVELOPMENT COST AND GAIN OF USAGE 77
6. CONCLUSION 91
REFERENCES 93

TABLES

Table 2.1: Creational Design Patterns..……………….……….……………………..……41

Table 2.2: Structural Design Patterns..…..……………..…..……………….……………………42

Table 2.3: Behavioral Design Patterns.……..……………...…………….………………………43

Table 3.1: Framework Cost Table..….………………………...……………………………46

Table 5.1: LOC Measurement for Sample Framework……...……………..……………………77

Table 5.2: LOC Measurement Based on Entity……………...……………………….…………78

Table 5.3: Unadjusted Function Point (UFP) Calculation…...…………………………….……80

Table 5.4: UFP Calculation for WebUtils.Authorization in the Case Study……...……….80

Table 5.5: UFP Calculation for Authorization Project in the Case Study……………...…81

Table 5.6: UFP Calculation for Data Project in the Case Study……………...……….....…81

Table 5.7: UFP Calculation for WebApiUtils Project in the Case Study……..……….....…81

Table 5.8: UFP Calculation for FTP Project in the Case Study……………...……………82

Table 5.9: UFP Calculation for E-Mail Project in the Case Study……………............…..82

Table 5.10: General System Characteristics…………..…………………...……...........…83

Table 5.11: General System Characteristics Scores for the Case Study………......………84

Table 5.12: Programming Language Statements Per Function Point……...……...........…85

Table 5.13: Calculated Lines of Code in the Framework Case Study..…...……...........…86

Table 5.14: Development Cost…………………………………..…..……...........…87

Table 5.15: SonarQube Analysis Report For Framework Case Study..…...……...........…89

FIGURES

Figure 2.1: Long parameter list…………………………………………………...........16

Figure 2.2: Use of classes instead of long parameter list……………………………...17

Figure 2.3: Indenting Code Sample…………………………………...……………...19

Figure 2.4: Error Handling With Flags…………………………………...…………….20

Figure 2.5: Error Handling With Exceptions...…………………………...…………...........21

Figure 2.6: Unit Test Sample……………………………..…………………....................23

Figure 2.7: Single Responsibility Principle Bad Example……………..……………...........27

Figure 2.8: Single Responsibility Principle Good Example………..…..…………….........28

Figure 2.9: Open Closed Principle Bad Example………………………...……….............30

Figure 2.10: Open Closed Principle Good Example……...………...………….................31

Figure 2.11: Liskov Substitution Principle Bad Example…..……………....……….................32

Figure 2.12: Liskov Substitution Principle Good Example………………....……………...33

Figure 2.13: Interface Segregation Principle Bad Example……………….........................35

Figure 2.14: Interface Segregation Principle Good Example………..……..……….............36

Figure 2.15: Dependency Inversion Principle Bad Example..………………..…..................38

Figure 2.16: Dependency Inversion Principle Good Example………..………………..........39

Figure 3.1: Token Based Authentication Life Cycle……………………………………….........51

Figure 3.2: Sql Injection Attack…………………………………………….......................52

Figure 4.1: BaseModel………………………..……………………………........................59

Figure 4.2: Linq Sample……………………..………….………………........................60

Figure 4.3: Repository Pattern with Unit of Work..………..………………….......................61

Figure 4.4: Use of LinqKit PredicateBuilder...…..…………………………..…...................62

Figure 4.5: Sample Use of Data Annotation………………..………………........................63

Figure 4.6: Sample Use of Fluent API…………..……………………………......................63

Figure 4.7: Permission Attribute Code…………..……………………….........................65

Figure 4.8: Sample Use of Permission Attribute...…………………………........................66

Figure 4.9: Convert Class..……………………..…………………………........................66

Figure 4.10: BaseController Class……………….……………………..………...............69
Figure 4.11: Login Method in TokenController Class……….……..………….............71
Figure 5.1: Reasons for Project Cancellation.……..…………….….…………….…..............74
Figure 5.2: LOC Estimated and LOC in Reality……….…………….…………..…..............86
Figure 5.3: SonarQube Report For Framework Case Study….….……………….................89
Figure 5.4: NDepend Report For Framework Case Study….….…………….....................90

ABBREVIATIONS

COCOMO	:	Constructive Cost Model
CRUD	:	Create, Read, Update, Delete
FP	:	Function Point
FPA	:	Function Point Analysis
FSM	:	Functional Size Measurement
HTTP	:	Hypertext Transfer Protocol
IDE	:	Integrated Development Environment
JSON	:	JavaScript Object Notation
KLOC	:	Thousands Of Lines Of Code
LINQ	:	Language Integrated Query
LOC	:	Lines of Code
ORM	:	Object Relational Mapping
REST	:	Representational State Transfer
RPC	:	Remote Procedure Call
SOAP	:	Simple Object Access Protocol
TDI	:	Total Degree Of Influence
UFP	:	Unadjusted Function Point
URI	:	Uniform Resource Identifier
VAF	:	Value Adjustment Factor
WEB	:	World Wide Web
WSDL	:	Web Services Description Language
XML	:	Extensible Markup Language

1. INTRODUCTION

Today, the rapid development of technology leads to an increase in competition. One of the most important requirements of the competition is to generate fast products. However, it is also very important to make no compromises of quality and security while generating fast outputs. Especially for companies that produce software, products with poor quality and security vulnerabilities cause serious reputation and financial loss. With the widespread use of the internet as well as computers in daily life, security has become a serious concern and security vulnerabilities are increasingly exploited by malicious or inexperienced users (Beydağlı, E., Kara, M., Bahşi, H., & Alparslan, E., 2010).

According to the data given in CHAOS report published by Standish Group in 2015 after conducting research on 50 thousand projects worldwide, the success rate of software development projects is only 29%. 52% of the projects are canceled and 19% of them result in failure. Once again, according to the data from 1999 report of CHAOS, the cost of unsuccessful projects is 75 billion USD. Total cost overrun is 22 billion USD (Rehber, D.).

There are many reasons for software development projects to result in failure, including but not limited to, insufficiently involved senior management, too many requirements and scope changes, lack of necessary management skills, and overbudget, etc. One of the most prominent reasons is development of non-expandable, inflexible software without quality or security measures being taken. According to a study conducted between 2005 and 2007, project cancellation ratio due to non-qualified software is approximately 11% (El Emam, K. & Koru, A. G., 2008).

Compliance with certain rules set by years of experience by various authorities during the development of software increases the quality and security of the product. In addition, the lifetime of the software product is significantly extended with certain fundamental upsides such as readability of code, flexibility, and extensibility.

For developers, it is possible to divide "learning code development" process into two main stages.

First stage consists of fundamental characteristics such as algorithm building, programming knowledge, and syntax specific to the language learned. It is necessary to achieve a certain level of experience at this stage in order to begin developing a software.

The second stage consists of software development techniques such as design templates, software development principles, safety rules, etc., which are recommended (non-mandatory) to be followed as a result of many years of experience. Software can also be developed without following these techniques, but there is a high probability to encounter security problems and various errors in the future. In addition, if some rules are not followed in order to develop flexible software, each change or fix of error will further complicate the software project and reduce readability of the code. After a while, a new phase of development will be needed for the complicated software project instead of intervention through updates and this will cause severe financial losses.

Each software developer's level of knowledge and coding technique differ from one another. Considering that companies have more than one software developer; development in compliance with certain standards and rules is important for the continuity of projects. When a software developer needs to work on a different project than the one they currently handle, although the flow and algorithms may be different, working on the project using the same rules and coding techniques will lead to shorter adaptation period to that project. This allows for increased productivity, higher quality, and safer and faster products.

Framework is a structure of software that provides developers with a variety of classes and functions to use in their projects. In companies where a wide variety of software is developed, development of software projects on the basis of a framework enables developers to follow certain rules and standards. As the basis of each project will be provided by the same framework (or frameworks), debugging, updates, and security enhancements are also provided in a centralized way as consistency is enhanced. When a new feature is to be added, any change to the framework will allow all projects to benefit from these features.

In this thesis study, the importance of utilizing from custom frameworks is emphasized for respective internal workflows at companies based on techniques of developing quality code within the company.

In the first chapter, under the title "software quality", frequent coding errors and certain software development principles recognized by authorities for developing quality software are investigated. In the second chapter, frameworks and the importance of their corporate use are provided; whereas the last chapter contains a sample framework study.

There are significant advantages of utilizing from a customized Framework product in respective workflows at companies that develop software, in terms of both code quality as well as costs.

In this thesis study, the advantages of utilizing from a Framework product that supports developing quality code as mentioned in the first chapter is investigated from the perspective of raising developer awareness in addition to encouraging code quality. As a result of this, more consistent and stable products can be ensured with extra flexibility in project outputs against new requests or modifications.

Likewise, centralizing common modules under the umbrella of a framework and saving time by utilizing from modules through frameworks instead of coding them from scratch in projects, are analyzed in Results chapter of the thesis along with related numerical data.

With the increase in the use of frameworks, software development processes can be shortened and the number of more consistent, stable, secure, and quality software can be increased, providing significant gains for both companies and the global economy.

2. CODE STANDARDIZATION AND SOFTWARE QUALITY

In this age, one of the most important factors for successful and long-lasting applications is proper coding. Having the "proper" code does not just mean that the application does the work right, but it also makes it easier to add new code, fixing, debugging, and maintaining the code. Maintenance of software projects constitute about 70% of the software's life cycle. Enhancing maintenance procedures ensures important advantages in terms of time and money for software developers (Razina, E., & Janzen, D. S., 2007).

2.1 CODE STANDARDIZATION

Coding standards are often overlooked, but they are of critical significance. A consistent coding standard helps to improve the quality of any software project. In order to talk about a consistent coding standard, the source code of the whole project will need to be harmonious as if it was coded by a single developer.

In consistent standardized coding, adaptation of a different developer to the project becomes much easier. This ensures that tasks such as new requests, editing or debugging required by different developers can be completed much more quickly and efficiently.

In order to follow a consistent standard, the developer will need to be aware of the quality coding standards that are accepted around the globe, as well as other company-specific standards where the developer works for. In addition, it is necessary to be aware of erroneous software development techniques referred to as "code smells" and such techniques should not be applied.

Quality coding standards (clean coding) and code smells are directly related to the developer's self-improvement and experience. Companies can also plan a specific training schedule in order for developers working within their organizations to master such issues.

For company-specific standards, companies need to prepare an easy-to-understand documentation and monitor compliance with these standards.

It is necessary to pay close attention to certain aspects while determining coding standards.

a) Readability of code
b) Separation of code blocks from one another
c) Using indentations at points where control structures such as *If else*, *switch case* begin and end
d) Determination of variable naming rules and ensuring such nomenclature describes the function of variable
e) Function nomenclature or methods to help understand what they do
f) Use of comments to inform about program flow at points where function or method nomenclature will not suffice
g) Avoiding complex structures that might have negative impact on readability
h) Getting feedback from team members on determination of standards

2.2 CODE SMELLS

The main purpose of software development is to meet customer expectations. Requests from customers are often complicated. On the contrary, customer expectation is a high quality product at the end of the day. The quality of the software to be developed in accordance with such expectations is critical

There are certain rules and principles set by specific authorities in developing quality software. However, when examining such rules and principles, software errors should not be ignored. Such errors are generally referred to as "code smells" in the world of software.

Code smell is the name given to pieces of code or designs that are likely to cause major problems in the future (Suresh, P. & MuthuKumaran, S., 2013). This term was first coined by Kent Beck and then referred to in Martin Fowler's book titled *Refactoring* for difficult code fragments and deformation in the universe of software. Martin Fowler underlines here that: "Code smells are indicators that will lead to further problems in a system".

Instead of codes that generate errors during compilation, the term "code smell" refers to codes that are functional with low readability and incompatible with accepted software development principles. In effect, code smells have a negative impact on the maintainability of software projects (Fontana, F. A., Dietrich, J., Walter, B., Yamashita, A., & Zanoni, M., 2016). A software developer who is not fully acknowledged on these details will not be aware that they develop a code that will cause major problems in the future as they will not see any errors during compilation (Kim, S., Whitehead Jr, E. J., & Zhang, Y., 2008). For this reason, it is imperative that such details are heeded by software developers.

The list of code Smells is described under 5 titles.

2.2.1 The Bloaters

The term *bloater* refers to code structures that grow without close control. This software problem does not reveal itself until further progress is made and someone divides the code upon realization.

These structures can be analyzed as follows:

Long Method: It is the case that appears by adding new codes to the existing method and giving the method different responsibilities in addition to its main purpose. The recommended or ideal method length is 10 lines. When this value is exceeded, readability of the code is reduced. Detection becomes difficult when a problem occurs, and fixing after detection is also problematic in long methods.

To avoid the problem, the method should be divided into small pieces as much as possible and each responsibility must be moved onto separate methods.

Large Class: These are the classes that contain various fields, methods and code lines, increasing complexity. They are usually designed to include small numbers of elements at first, but then enlarge in direct proportion to the application, leading to complexity. As a solution, classes can be fragmented, subclasses or interfaces can be used, as well. Dividing classes prevents repetition of problems in the code.

Primitive Obsession: It refers to representing complex elements with primitive data types.

For example, instead of designing a specific object to hold a phone number value, they can be represented as an integer type or values defined as enums can be kept constant.

In the first example, designing an object to hold a phone number value is more difficult than using a direct integer value, but it provides many advantages when using operations specific to phone number values. For instance, the prefix field can be restricted with a dropdown list that retrieves the source from the enum so that certain values can be entered; however, if it is defined as an integer, the user can enter any value they like into the prefix field.

In the second example, using an enum instead of a constant is useful in restricting and listing values. Consequently, it is possible to make the code more flexible by avoiding primitive obsession, making it more legible and safer against bugs.

Long Parameter List: It refers to the problem that appears consequent to the use of three or more parameters in one method. The use of excessive parameters reduces readability of the code, meanwhile increasing its dependency on objects calling it. For example, when a new parameter needs to be added, it will be necessary to make this change also in the lines where it is called.

As a solution, parameters can be brought together under a class. This will reduce dependency. In this case, readability of the code will increase while length of the code decreases.

Data Clumps: A code block may be needed in several sections. In such cases, if the same code block is copied to each required field as needed (copy-paste programming), it turns into an inconvenience. It may also cause unnecessary redundancy in the code, and should be changed in all copied sections when a single change is required in this code block.

Instead, the code block to be repeated must be defined in a separate method or class, and this method should be called where it is needed. In this way, code redundancy can be prevented while readability and maintenance of the code becomes much easier.

2.2.1 Object Oriented Abusers

This term refers to object oriented programming principles that are incomplete or used incorrectly.

Switch Statements: Structures such as *switch* and *if-else*, specifically used in factory templates, seem useful but reduce code readability and flexibility when used unnecessarily. When a new condition is to be added, it will be necessary to find out where the switch structure is used within the code, and add the new condition there.

In order to reduce problems caused by the use of these structures, the code in which the structure is used can be isolated in a separate method, or polymorphism can be used if the structure allows it.

Polymorphism: It is the term given to the use of the same class member by objects implementing the same interface or derived from the same abstract class.

Temporary Fields: Temporary fields retrieve values in certain conditions required by objects, and in other conditions the values are empty. To read and understand a code consisting of such values is difficult as it requires to keep track under which conditions the field is empty or not.

As a solution, these values can be used in separate classes or methods depending on the condition, or null-object fields can be used, as well.

Refused Bequest: It is the term given to a sub-class using certain methods it inherits from the class it is dependent on. In this case, other methods will become useless and errors will have to be thrown from these methods.

In such an event, it is highly probable that either the structure is constructed incorrectly, or the upper class and the lower class become distinct from each other in time.

To eliminate this problem, if the relationship between the lower class and the upper class becomes irrational, it is necessary to remove this relationship. If this relationship is reasonable but there are still fields that should not be used in the subclass, then gathering this method or fields in a separate class or generating a new interface and then implementing the sub-class from this interface may be the option to overcome this problem.

Alternative Classes with Different Interfaces: This refers to the event that two separate classes have different method names with methods that perform the same function. Generally, this is caused when the programmer forgets that they had generated such a class, or when the programmer is not aware of the class generated by another developer.

To resolve the problem, naming can be changed in the same way or similar methods can be gathered under the same class. If only certain parts of the classes are similar, then these methods can be combined in a different upper class and other classes may inherit from that class. Whichever solution is preferred, it should be remembered to delete methods at first.
In that way, the problem of duplicated code will be avoided and the readability and flexibility of the code will increase.

2.2.2 Change Preventers

Whenever a change is made on a project, it is undesirable to necessitate the change on many other fields, as well. As a result, the project gets complicated and the readability becomes difficult.

Divergent Change: This refers to the event when a change is made in a class and further changes are required on many other methods. It is often a result of poor design or copy-paste programming. As a solution, code repetition should be reduced.

Shotgun Surgery: This is similar to divergent change, but the difference is that divergent surgery is related to a change made only in one class. Shotgun surgery on the other hand occurs when a change necessitates further changes in various classes.

It occurs when a responsibility is divided into various classes. As a solution, methods need to be collected under the same class.

Parallel Inheritance Hierarchies: This refers to the event when a sub-class is created for a class, another sub-class is necessary to be created for other classes, too. Incorrect design of the hierarchy or growth of hierarchy at an unpredictable level in the design phase leads to this problem.

2.2.3 Dispensables

All elements that make the code less readable and difficult to understand fall into this category.

Comments: Although comment lines actually seem to be quite useful, it is necessary to use this feature at the minimum. They are often added to make a difficult-to-understand code more understandable. In such cases, instead of adding a comment line, first choice should be writing the difficult-to-understand code in a simpler manner. Common opinion on this indicates that: "The best comment is a good name for a method or class."

Duplicate Code: This occurs due to repetition of code. It can be seen mostly because of human factor rather than design. Duplicate code occurs intentionally for reasons such as laziness or incompetency of the developer, or even the attempt to write the code faster in order to meet a given deadline. In addition, duplicate code can also be caused by developers working on the same project as they code similar functionalities unaware of each other.

Code repetition is a significant code smell that results in poor design. It leads to increase in the occurrence and possibility of propagation of errors, consequently, greatly increasing maintenance costs (Bulychev, P., & Minea, M., 2008).

Code fragments that fulfill the same function can be gathered under a single method or they can be combined in a separate class if there are methods that fulfill the same function in different classes.

Lazy Class: Classes must be readable and flexible at the same time in order to ensure that later adjustment is possible. This, too, has a high cost. Therefore, unnecessary classes that do not have any function should be deleted. Typically, such classes are coded with certain responsibilities during the design phase, but in time via updates and refactoring, responsibilities are reduced and shrink significantly. Such classes must either be deleted or moved to classes that have similar responsibilities.

Data Class: It is the event when a class only consists of fields and getters/setters. It is recommended to also have methods that perform certain functionalities.

Dead Code: This expression is used for variables, fields, methods, or classes that are no longer used. Generally, this happens when no one cleans the code after modifications in the software project. Such codes should be deleted when found. Such codes can be easily detected by use of advanced IDEs at present.

Speculative Generality: This refers to generation of empty or non-functional classes, methods or parameters that are predicted to be used in the future. This reduces readability of the code and makes it difficult to understand the flow. By deleting such structures, the code becomes more readable, simpler and easier to maintain.

2.2.4 Couplers

Problems that arise as a result of over-dependence between classes fall into this category.

Feature Envy: This is the term given to the event when a method accesses data more than its master. In this case, the structure must be examined and the code piece that accesses the method or data must be migrated if necessary.

Inappropriate Intimacy: It is the term given to a class's access to internal fields or methods of another class. Ideally, a class should have very little information about the other classes. Therefore, class elements that are unnecessary to open externally should be encapsulated. If there is an error in the structure, the relationship between the classes should be re-examined.

Message Chains: This is the status where method calls occur in chain. In such cases, editing the code becomes very costly due to the tight dependency of objects. Interclass dependence needs to be reduced.

Middle Man: This is the term given to the creation of a class only for the purpose of running another class. It would be useful to migrate the class to the other.

2.3 CLEAN CODING

Sloppy code is a frequently encountered problem due to lack of knowledge or experience, or may be related to concerns on finishing the project on time.

If the code is sloppy due to lack of knowledge or experience rather than delivering the project on time, the general consensus is that the code will be re-edited at the “first possible opportunity”. But this is an incorrect approach. Considering LeBlanc's "later equals never" rule, probably that "first possible opportunity" will never come due to edits or new requests related to the project or due to multiple projects.

In a project that starts with sloppy code, the code becomes complicated after new features or edits are added over time. This makes the code difficult to read and to update. In addition, the efficiency of projects that require teamwork will be reduced. In some projects, this becomes very complicated and it even causes the project to fail.

The only way to prevent this is clean coding. It requires certain basic coding rules and performing the job with discipline.

2.3.1 Naming Convention

The names selected for variables, methods, and classes is crucial. For a more understandable and clean coding, certain rules must be considered while naming them.

The function of the element (variable, method, class, etc.) to be named must be understood from its name.

int a; // Incorrect use
int countOfCustomer; //Correct use

In these two code fragments, there is no information for the variable *a*. This is an incompatible way regarding the clean code principles, as there is no information in the lower parts of the code, even if its function is adjacently given as in comment line.

In the code fragment below, it can clearly be understood from its naming that it is a variable indicating the number of customers.

Unnecessary abbreviations should be avoided

int cOfCust; //Incorrect use

It cannot be understood that *c* is an abbreviation of count, and *Cust* is an abbreviation of customer. Therefore, abbreviating in the way that other developers may not be able to understand, should definitely be avoided.

Names should be easy to pronounce

int cntCstmr; //Incorrect use

The use of such names is quite common in order to shorten variable names comprehensibly, but this is not a convenient way of using clean code principles. Naming must be clear and easy to pronounce.

Meaningful naming should be used also for constants

```
for (int k = 0; k < 12; k++) //Incorrect use
{
    y++;
}
for(int k = 0; k < NUMBER_OF_MONTH_PER_YEAR; k++) //Correct use
{
    countOfYear++;
}
```

In the above code fragment, the condition for *k* variable is to be smaller than 12; however, there is not a single hint indicating why 12 is used. Instead, use of a constant variable as below should be preferred in terms of readability and comprehension of code.

Class names must be nouns or noun-phrases. They should not contain verbs

Classes should be named as noun phrases such as *Customer*, *FileParser,* which indicate their function.

Method names must be either verbs or verb phrases.

Methods are code fragments that execute a certain job. Therefore, while choosing a name, phrases should be chosen as to indicate their function. These expressions must be either verbs or verb phrases.

E.g.: *GetCustomerById, DeleteFile* etc.

Concepts should be predetermined.

Particularly in team work, names of objects to be used should be determined in advance and team members should be informed about it. Otherwise, generation of multiple objects with the same function would occur. This leads to a duplicate-code situation and causes code complexity for the future.

As an example, if data is going to be received from a web service, name of this service should be the same everywhere. If one team member uses the expression of WebApi for web service, and the other one uses webService, no healthy listing can be made about the methods related to this service.

2.3.2 Functions & Methods

The functions or methods should be short, understandable, and their intended functions should be obvious. Depending on the Single Responsibility principle, a method or function must have a single task (responsibility).

"The first rule of functions is that they should be small. The second rule of functions is that they should be smaller than that"

The rule that was common in the 1980s was that only one function would fit on the screen at most. In those days, approximately 24 lines could fit on the screen. This rule has lost its validity because today's monitors are much bigger. Today, a function should not exceed approximately 20 lines.

Name of functions or methods should be clear, and provide an idea of what the function does.

If functions or methods have more than 3 or 4 parameters like at the Figure 2.1, such parameters should be gathered inside a class (Figure.2.2) . Thus, readability and usability would increase.

Figure 2.1: Long parameter list

```
public bool Create(string userName, string password, string email, string name,
string surname)
{
    //...
}
```

Figure 2.2: Use of class instead of long parameter

```
public class UserRequest
{
    public string UserName { get; set; }
    public string Password { get; set; }

    public string Email { get; set; }
    public string Name { get; set; }
    public string Surname { get; set; }
}

public bool Create(UserRequest userRequest)
{
    //...
}
```

2.3.3 Comments

Instead of writing comments, clean and clear coding should be preferred to prevent the need for a comment. Readability of a clean code can be increased by naming understandably. Such a code usually does not require any comment.

There is a huge drawback in using comments, which is the inability to update comments despite the code change. In such a case, comment lines may provide incorrect information.

Codes written in the past, which are no longer needed or are not expected to be used, should not be translated into comments. It is unnecessary to translate codes that need to be deleted, since they would already be stored in version control tools to be examined later, it also decreases readability of the code.

There are times when comments need to be added. Examples are legal comments for programmers, warning comments for other developers or for people reading the code, and TODO comments for future work.

As for comments considered to be required, expressions must be clear and descriptive. The reader should have an idea about the code from the comments.

2.3.4 Formatting

While coding, formatting the text greatly increases readability of the code. For instance, beginning the code within *if* or *for* conditions indented with certain amount of space indicates that code block is located within a condition. This method is referred to as indentation. Almost all of current advanced IDEs perform indentation automatically.

In addition, another point to note when formatting is the fact that similar concepts are close to one another. For example, there should be no line spacing between lines within the method (assuming the method is to be kept short, otherwise readability will decrease again) but there should be a certain amount of space between the two methods.

Variables must be defined at a position near the code block they will be used. For example, the variable to be used in a *for* loop should be defined immediately above the *for* block.

If there are methods or functions that call each other; these methods or functions within the class should be close to one another. Thus, tracking the code can be performed more rapidly and comfortably.

Line length is another formatting method that significantly affects code readability. Tracking the lines that do not fit in the screen is quite difficult. The maximum line length accepted by authorities is 120 characters. Current advanced IDEs automatically adjust the line length to the screen size.

Figure 2.3: Indenting Code Sample

```
protected override async Task HandleRequirementAsync(AuthorizationHandlerContext
context
                                    , PermissionAuthorizationRequirement
                                    requirement
                                    , IEnumerable<PermissionAttribute> attributes)
{
    var claimsPrincipal = context.User;
    UserIsAuthenticated(claimsPrincipal);
    try
    {
        var userId = _apiRequest.User.Id;
        var roles = (await
_userRoleManager.GetRolesByUserIdAsync(userId)).EntityList;

        foreach (var permissionAttribute in attributes)
        {
            if (permissionAttribute.CustomClaim != null)
            {
                await CheckCustomClaimAsync(userId
                            , roles
                            , (CustomClaim)permissionAttribute.CustomClaim);
            }
            else if (permissionAttribute.Entity != null &&
permissionAttribute.Crud != null)
            {
                await CheckEntityClaimAsync(userId
                                          , roles
                                          , (Entity)permissionAttribute.Entity

            }
            else
            {
                throw new UnauthorizedAccessException ();
            }
        }
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
    context.Succeed(requirement);
}
```

2.3.5 Error Handling

Things may not always function as expected in applications. Errors or bugs may occur in the application, be it expected or unexpected. Managing such errors is one of the crucial requirements of clean coding.

The use of flags is a common procedure for error handling (Figure 2.4). A constant is set for each error scenario and the constant value is returned according to it. It is necessary to input comments for these returned values in the same place or in a different place. However, this method requires control over all flags or constants, making it more difficult to control as the application expands.

Figure 2.4: Error Handling with Flags

```
public enum Error
{
    LoginError,
    ConnectionError,
    UnknownError
}

public class GenericResponse
{
    public bool Success { get; set; }
    public Error Error { get; set; }
}

public GenericResponse Login(string username, string password)
{
    var response = new GenericResponse();
    if (!_user.CheckUser(username, password))
    {
        response.Success = false;
        response.Error = Error.LoginError;
    }

    //...

    return response;
}
```

It is a better choice to throw an error by using exception objects instead of this method (Figure 2.5). In this way, the code can be separated and tracking the code becomes easier. The code block to be written for this must be enclosed in a try-catch-finally block. If the exception is to be thrown from a different method, the method should be called in a try-catch block.

Figure 2.5: Error Handling with Exceptions

```
public bool Login(string username, string password)
{
    try
    {
        if (!_user.CheckUser(username, password))
            throw new AuthenticationException();
        //...
        return true;
    }
    catch (Exception)
    {
       throw;
    }
}
```

As for exceptions, the error text should be clear. It should provide information about the process that caused the error. If logging is performed, these errors should also be logged with clear information in the catch block.

To minimize unexpected errors, it is necessary to use null values as little as possible. For example, a method that can return a null value is potentially a cause of error. Similarly, a method that accepts null values may generate errors according to the scenario.

2.3.6 Unit Tests

Testing is the process of validating whether a system meets specified requirements, or determination of differences between the expected and the observed results by way of trying or evaluating it manually or automatically (Tiftik, N., Öztarak, H., Ercek, G., & Özgün, S., 2007).

Unit test, on the other hand, is the test of software units (class, method, function, etc.) within the system. What is important in unit tests is testing only one feature at a time, and in that way, it becomes easy to understand what is to be tested or where the problem is (Mackinnon, T., Freeman, S., & Craig, P., 2000). It is checked whether these units produce the expected result for the determined inputs or not.

Coding a unit test reduces errors and the developer is forced to write quality code. The most important rule of unit testing is to isolate the unit to be tested from other components it is dependent on.

For example, if a database operation is being performed on the unit to be tested, this must be performed in the unit that is associated with the interface. Otherwise, the relevant unit becomes dependent on the database component.

While coding the unit test for this process, a mock object of the database component is returned according to the scenario to be determined by the developer. The unit test also checks whether the method returns the value according to this scenario or not.

The same operation cannot be performed if there are dependent components in the system. In this case, the developer, adding a unit test to the project will reduce the dependency on its components compulsorily. This increases both code quality and code readability for other developers.

Unit test methods generally use the AAA (Arrange, Act, Assert) pattern. According to this pattern, at the beginning of the method, necessary arrangements are made for the test scenario. In this part, the data set to be used in the method to be tested is set according to the test scenario. This part is referred to as Arrange.

Figure 2.6: Unit Test Sample

```
[TestClass]
public class DbContextTests
{
    private IEnumerable<Customer> customers { get; set; }

    [TestInitialize]
    public void Setup()
    {
        var stream = NorthwindData.CustomerData.ToStream();
        var reader = new StreamReader(stream);
        var serializer = new XmlSerializer(typeof(Collection<Customer>));
        customers = (Collection<Customer>)serializer.Deserialize(reader);
    }

    [TestMethod]
    public void ConfirmDbContextUsesDummyData()
    {
        var mockContext = new Mock<INorthwindContext>(); // Arrange
        var customerDbSet = customers.GetQueryableMockDbSet(); //Populate Customers "table"
        foreach (var customer in customers)
        {
            customerDbSet.Add(customer);
        }
        mockContext.Setup(context => context.Customers).Returns(customerDbSet);
        var dbContext = mockContext.Object;
        var customerData = dbContext.Customers; // Act
        Assert.IsTrue(customerData.Count() == 3); // Assert
    }

    [TestMethod]
    public void ConfirmThatRepositoryUsesFakeCustomerData()
    {
        var mockContext = new Mock<INorthwindContext>(); // Arrange
        var customerDbSet = customers.GetQueryableMockDbSet(); //Populate Customers "table"
        foreach (var customer in customers)
        {
            customerDbSet.Add(customer);
        }
        mockContext.Setup(context => context.Customers).Returns(customerDbSet);
        var dbContext = mockContext.Object;

        var mockRepository = new Mock<ICustomerRepository>();
        mockRepository.Setup(e => e.GetAll()).Returns(dbContext.Customers);
        var customerRepository = mockRepository.Object;

        var records = customerRepository.GetAll(); // Act
        Assert.IsTrue(records.Count() == 3); // Assert
    }
}
```

Source: https://www.danylkoweb.com/Blog/the-fastest-way-to-mock-a-database-for-unit-testing-B6

In the Act part, the method to be tested is called along with the parameters set during Arrange. In Assert, it is checked whether the expected result returns from the tested method or not. According to this part, the test method succeeds (Passed) or generates error (Failed). For consistency of the system, it is not necessary to deploy until all unit tests are successful. According to Roy Osherove's book titled *The Art of Unit Testing*, unit tests are:

a) Automated and repetitive
b) Simple to implement
c) Maintainable
d) Everyone can operate unit tests
e) Run fast
f) Run in memory

In order to apply a unit test:

a) Tests should be easily operated and they should be cleansed of code crowds
b) No hand-written Stub/Mock objects should be used
c) Code should be repetitive
d) Multiple assert should not be used
e) Only one mock object should be used per test

2.3.7 Classes

When designing a class, sorting the elements in the class is important. Sorting increases readability of the code and it also helps protect a particular discipline in interventions that will be needed later in the code.

Classes, like methods, should be as small as possible. Their responsibilities must be clear and the responsibilities must be understood from the class name.

The accepted, ideal design principles for classes are investigated under solid principles in this thesis study.

2.4 SOLID PRINCIPLES

Solid principles consist of five principles suggested by Robert C. Martin in early 2000s. The initials of these five principles constitute the word SOLID. The main objective of these principles is to improve the Separation of Concerns through High Cohesion and Low Coupling.

Separation of Concerns: It is the term given to the division of concepts that are not related to one another. Multi-layer architecture can be given as an example. (Data Layer, Presentation Layer etc.)

Low Coupling: Dependency between classes is referred to as coupling. There are two sub-concepts here. These are loose coupling or low coupling, and tight coupling.

Tight coupling occurs in the following cases:

a) *B-class member (reference, pointer, or object) exists in class A*
b) *Objects of class A call methods of class B*
c) *B-class member (reference, pointer, or object) is either received or returned by a method of class A as a parameter*
d) *Class A method has a local variable from class B*
e) *Class A is the sub-class of class B*

Higher dependency increases maintainability, understandability, and reusability, so the ideal approach should be loose coupling (Ural, E., Tekin, U., & Buzluca, F., 2008).

High Cohesion: Cohesion is the common use of data in a class and methods to implement the same purpose. Methods should perform one task, but methods in the same class must serve for a common purpose. Therefore, there should be no methods irrelevant to each other within a class. In effect, in the ideal development environment, cohesion should be as high as possible.

Low cohesion leads to the following problems:

a) *Understandability*
b) *Maintainability*
c) *Reusability*
d) *Fragility (Ural, E., Tekin, U., & Buzluca, F., 2008)*

For systems that are rigid, fragile and stable, it is very difficult to change existing functionality or add new features.

Rigidity: *This refers to designs that are not update-friendly.*
Fragility: *This refers to any change that is required to be made causing problems in another part of the project.*
Immobility: *This refers to the developed modules not being suitable for reuse.*

In systems with poor dependency management, a chain of mistakes may occur inevitably when a function is changed. For this reason, utilizing from SOLID principles contributes towards more efficient, reusable, and non-fragile software development. This, in turn, ensures development of projects with higher long-term sustainability and maintainability.

2.4.1 Single Responsibility Principle

In this principle, the goal is to have a single responsibility for code units, such as a class or a method. Multiple responsibilities increase complexity of code units in extreme and unnecessary ways. A complex code's fragility is high, whereas its flexibility is low. The reason being it has more than one responsibility, and changes will affect other lines, which also increases immobility.

A structure with a single responsibility is both small in size and easy to manage. In addition, testability of the system increases, meanwhile decreasing the number of unit tests (Özçelik, O., & Altilar, D. T., 2015).

Figure 2.7: Single Responsibility Principle – Bad Example

```
public class LoginService
{
    private readonly IUserRepository _userRepository;

    public LoginService(IUserRepository userRepository)
    {
        _userRepository = userRepository;
    }

    public bool Login(string email, string password)
    {

        if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(password))
        {
            throw new Exception("Bad request");
        }

        if (_userRepository.GetUserByEmailAndPassword(email, password) == true)
        {
            return true;
        }
        else
        {
            var smtpClient = new SmtpClient();
            smtpClient.Send(new MailMessage("admin@sample.com", email) { Subject =
"Login error" });
            return false;
        }
    }
}
```

The example in the Figure 2.7 indicates a situation that is incompatible with the Single Responsibility principle.

The login method assumes multiple responsibilities, such as validation and sending an e-mail, as well as invoking the user authenticating method, which is a normal task.

Figure 2.8: Single Responsibility Principle – Good Example

```
public interface IUserValidation
{
    void Validate(string email, string password);
}

public interface IEmailService
{
    void SendEmail(string from, string to, string title);
}

public class UserValidation : IUserValidation
{
    public void Validate(string email, string password)
    {
        if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(password))
            throw new Exception("Bad request");
    }
}

public class EmailService : IEmailService
{
    public void SendEmail(string from, string to, string title)
    {
        var smtpClient = new SmtpClient();
        smtpClient.Send(new MailMessage(from, to) { Subject = title });
    }
}

public class LoginService
{
    private readonly IUserRepository _userRepository;
    private readonly IUserValidation _userValidation;
    private readonly IEmailService _emailService;

    public LoginService(IUserRepository userRepository, IUserValidation
userValidation, IEmailService emailService)
    {
        _userRepository = userRepository;
        _userValidation = userValidation;
        _emailService = emailService;
    }

    public bool Login(string email, string password)
    {
        _userValidation.Validate(email, password);
        if (_userRepository.GetUserByEmailAndPassword(email, password) == true)
            return true;
        else
        {
            _emailService.SendEmail("admin@sample.com", email, "Login Error");
            return false;
        }
    }
}
```

In the Figure 2.8, responsibilities are migrated to different classes and methods, and interfaces of fragmented classes are created due to the Separation Of Concerns, and interfaces of these classes are added to the LoginService class instead of the EmailService and UserValidation classes. In this way, there is no dependency of the LoginService class to the EmailService or UserValidation classes. (Low-Coupling)

2.4.1 Open/Closed Principle

In this principle, the purpose is to design objects closed for modification but open for extension (Chebanyuk, E., & Markov, K., 2016).

In codes that do not utilize this principle, *if*s or *switch* structures are generally used; and while adding a new property, the new code is added to these *if* or *switch* structures. This makes the application open for modification.

Example code in the Figure 2.9 is a poor design. In this example that calculates area, every new shape added is coded in the *if* block. Therefore, when a new shape is to be added, a new *if* block will have to be added and in this case, the "closed for modification" principle would be violated.

When this principle is applied correctly, a base class is generally defined and this class is not changed at all, but instead new classes derived from this class are added in while the required properties are added to the application. In this way, development of the design can be cleaner, but it becomes closed for modification.

Figure 2.9: Open/Closed Principle - Bad Example

```
public class Rectangle
{
    public double Width { get; set; }
    public double Height { get; set; }
}

public class Circle
{
    public double Radius { get; set; }
}


public class CombinedAreaCalculator
{
    public double Area(object[] shapes)
    {
        double area = 0;
        foreach (var shape in shapes)
        {
            if (shape is Rectangle)
            {
                Rectangle rectangle = (Rectangle)shape;
                area += rectangle.Width * rectangle.Height;
            }

            if (shape is Circle)
            {
                Circle circle = (Circle)shape;
                area += (circle.Radius * circle.Radius) * Math.PI;
            }
        }
        return area;
    }
}
```

Source: https://exceptionnotfound.net/simply-solid-the-open-closed-principle/

In the Figure 2.10, a version of the previous poor example is given in compliance with the Open/Closed principle. Each shape to be added is derived from an abstract class named Shape, and this abstract class is used in the code block that calculates the field. Only new classes will be created for new shapes to be added (Open for Extension principle), but no modification will be made on code calculating area (Closed for Modification principle).

Figure 2.10: Open/Closed Principle - Good Example

```
public abstract class Shape
{
    public abstract double Area();
}

public class Rectangle : Shape
{
    public double Width { get; set; }
    public double Height { get; set; }
    public override double Area()
    {
        return Width * Height;
    }
}

public class Circle : Shape
{
    public double Radius { get; set; }
    public override double Area()
    {
        return Radius * Radius * Math.PI;
    }
}

public class Triangle : Shape
{
    public double Height { get; set; }
    public double Width { get; set; }
    public override double Area()
    {
        return Height * Width * 0.5;
    }
}

public class CombinedAreaCalculator
{
    public double Area(Shape[] shapes)
    {
        double area = 0;
        foreach (var shape in shapes)
        {
            area += shape.Area();
        }
        return area;
    }
}
```

Source: https://exceptionnotfound.net/simply-solid-the-open-closed-principle/

2.4.2 Liskov Substitution Principle

According to this principle, objects generated from sub-classes must display the same behavior when they are replaced with objects of upper classes.

For this, a sub-class must use all the properties of the upper class it inherits. If there are one or more properties that will not be used, this means that there is a problem in the design. Properties that are not usually utilized to solve this design problem should be migrated to a new interface or several interfaces. Sub-classes that utilize from these properties should define this interface with the upper class that it is supposed to inherit, while sub-classes that do not utilize from these properties should only inherit from the upper class (Liskov, B., 1988).

Figure 2.11: Liskov Substitution Principle Bad Example

```
public abstract class Bird
{
    public abstract void Fly();
}

public class Crow : Bird
{
    public override void Fly()
    {
        throw new NotImplementedException();
    }
}

public class Ostrich : Bird
{
    public override void Fly()
    {
        throw new NotImplementedException();
    }
}
```

Source: https://sanaulla.info/2011/11/28/solid-liskov-substitution-principle-2/

In the Figure 2.11, Crow and Ostrich classes are derived from the Bird class with the Fly method, but since Ostrich is a non-flying bird, Fly method will not be available. In this case it will not be able to exhibit the same behavior as the Bird class. Therefore, the design provided in the example is an incorrect design according to the Liskov Substitution Principle.

Figure 2.12: Liskov Substitution Principle Good Example

```
public abstract class Bird
{
    public abstract void Eat();
}

public abstract class FlightBird : Bird
{
    public abstract void Fly();
}

public abstract class NonFlightBird : Bird
{

}

public class Crow : FlightBird
{
    public override void Eat()
    {
        throw new NotImplementedException();
    }

    public override void Fly()
    {
        throw new NotImplementedException();
    }
}

public class Ostrich : NonFlightBird
{
    public override void Eat()
    {
        throw new NotImplementedException();
    }
}
```

Source: https://sanaulla.info/2011/11/28/solid-liskov-substitution-principle-2/

The code block in the Figure 2.12, the previous incorrect design of the example is corrected via creating new abstract classes. Crow and Ostrich classes are derived from newly created abstract classes, and the difference in behavior between the lower class and the upper class is removed in this way. As a second method, instead of creating new abstract classes, interfaces for Fly property could be created and sub-classes could implement this interface, as well.

2.4.3 Interface Segregation Principle

According to this principle, objects should not be forced to implement the interfaces that contain methods or properties they do not require. Such cases indicate that something is wrong with the design. Due to the need for more methods or properties, rather than using a single, large interface, multiple interfaces should be preferred in accordance with High-Cohesion principles where interrelated methods are involved (Sjøberg, D. I., Yamashita, A., Anda, B. C., Mockus, A., & Dybå, T., 2013).

In the Figure 2.13, some properties that a smartphone should implement are defined in the ISmartPhoneFeatures interface. The problem with this design is that these methods are unnecessarily implemented since there is no InfraRed receiver or Wi-Fi on every single phone. Instead, it would be a more accurate design by dividing the ISmartPhoneFeature interface and implementing the necessary interface.

Figure 2.13: Interface Segregation Bad Example

```
public interface ISmartPhoneFeatures
{
    void TurnOnDevice();
    void TurnOffDevice();
    void Call();
    void SendSms();
    void TurnOnInfraRed();
    void TurnOffInfraRed();
    void TurnOnWifi();
    void TurnOffWifi();
}

public class SmartPhoneModel1 : ISmartPhoneFeatures
{
    public void TurnOnDevice()
    {
        //Turn on device
    }

    public void TurnOffDevice()
    {
        //Turn off device
    }

    public void Call()
    {
        //Call
    }

    public void SendSms()
    {
        //Send SMS
    }

    public void TurnOnInfraRed()
    {
        throw new NotImplementedException();
    }

    public void TurnOffInfraRed()
    {
        throw new NotImplementedException();
    }

    public void TurnOnWifi()
    {
        //Turn on wi-fi
    }

    public void TurnOffWifi()
    {
        //Turn off wi-fi
    }
}
```

Figure 2.14: Interface Segregation Good Example

```
public interface ISmartPhoneBaseFeatures
{
    void TurnOnDevice();
    void TurnOffDevice();
    void Call();
    void SendSms();
}

public interface IInfraRedFeatures
{
    void TurnOnInfraRed();
    void TurnOffInfraRed();
}

public interface IWifiFeatures
{
    void TurnOnWifi();
    void TurnOffWifi();
}

public class SmartPhoneModel1 : ISmartPhoneBaseFeatures
{
    public void TurnOnDevice()
    {
        //Turn on device
    }

    public void TurnOffDevice()
    {
        //Turn off device
    }

    public void Call()
    {
        //Call
    }

    public void SendSms()
    {
        //Send SMS
    }
}
```

```
public class SmartPhoneModel2 : ISmartPhoneBaseFeatures, IWifiFeatures
{
    public void TurnOnDevice()
    {
        //Turn on device
    }

    public void TurnOffDevice()
    {
        //Turn off device
    }

    public void Call()
    {
        //Call
    }

    public void SendSms()
    {
        //Send SMS
    }

    public void TurnOnWifi()
    {
        //Turn on wi-fi
    }

    public void TurnOffWifi()
    {
        //Turn off wi-fi
    }
}
```

The Code block in the Figure 2.14, it can be seen that refactoring was performed on the previous incorrect design. Smartphone features are designed in three separate interfaces as basic features, infrared feature and WiFi feature; in this way, SmartPhoneModel1 and SmartPhoneModel2 classes implement interfaces that are compatible according to hardware specifications.

2.4.4 Dependency Inversion Principle

According to this principle, higher-level classes should not be dependent on lower-level classes. The relationship between them must be provided using abstraction or an interface (Singh, H., & Hassan, S. I., 2015).

If this principle is not complied with, a modification in sub-classes will also affect the upper classes. This causes complication in the code. At the same time, this also prevents reusability.

Figure 2.15: Dependency Inversion Bad Example

```
public class Email
{
    public string ToAddress { get; set; }
    public string Subject { get; set; }
    public string Content { get; set; }
    public void SendEmail()
    {
        //Send email
    }
}

public class Sms
{
    public string PhoneNumber { get; set; }
    public string Message { get; set; }
    public void SendSms()
    {
        //Send sms
    }
}

public class Notification
{
    private readonly Email _email;
    private readonly Sms _sms;
    public Notification()
    {
        _email = new Email();
        _sms = new Sms();
    }

    public void Send()
    {
        _email.SendEmail();
        _sms.SendSms();
    }
}
```

Source: https://exceptionnotfound.net/simply-solid-the-dependency-inversion-principle/

In the Figure 2.15, Notification class is highly dependent on E-mail and SMS classes. This also is against the Low-Coupling principle.

Figure 2.16: Dependency Inversion Good Example

```
public interface IMessage
{
    void SendMessage();
}

public class Email : IMessage
{
    public string ToAddress { get; set; }
    public string Subject { get; set; }
    public string Content { get; set; }
    public void SendMessage()
    {
        //Send email
    }
}

public class Sms : IMessage
{
    public string PhoneNumber { get; set; }
    public string Message { get; set; }
    public void SendMessage()
    {
        //Send sms
    }
}

public class Notification
{
    private readonly ICollection<IMessage> _messages;

    public Notification(ICollection<IMessage> messages)
    {
        this._messages = messages;
    }
    public void Send()
    {
        foreach (var message in _messages)
        {
            message.SendMessage();
        }
    }
}
```

Source: https://exceptionnotfound.net/simply-solid-the-dependency-inversion-principle/

The code block in the Figure 2.16, both the SMS and Email classes implement this interface named IMessage for sending messages. The Notification class also only recognizes the IMessage interface. This design eliminates the strict dependency problem of the Notification class on both the Email and SMS classes.

2.4.5 Other Principles

There are other principles in addition to the Solid Principles.

Kiss Principle (Keep it simple): If there is more than one option to solve a problem, the simplest and most understandable one should be chosen.

Yagni Principle (You Aren't Gonna Need It): According to this principle, code that is not needed should not be used. Additional enhancements should not be made on assumptions about properties that may be added in the future (Coram, M., & Bohner, S., 2005).

Dry Principle (Don't Repeat Yourself): Using the same code in different places in a project should be avoided. Code repetition will complicate the project, but at the same time it will need to make this modification in every copy when a modification is requested at the same time. This will not only increase coding time unnecessarily but also make the project more vulnerable to the possibility of being forgotten (Hunt, A., & Thomas, D., 2002).

2.5 DESIGN PATTERNS

Design Patterns are patterns that have been developed by authorities from the past to the present, as a solution to certain problems experienced. Christopher Alexander elaborates that: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" (Vlissides, J., Helm, R., Johnson, R., & Gamma, E., 1995).

The design patterns started in the end of the 1970s became popular with the book "Design Patterns" authored by Eric Gamma, Richard Helm, Ralph Johnson, and John Vlissides in 1994, also known as the Gang of Four, afterwards becoming crucial in the world of software. In this book, 23 patterns are explained in 3 different categories

Creational Patterns: Creational Design Patterns are designs used to create objects. They make the object creation process more efficient and flexible (Antoniol, G., Fiutem, R., & Cristoforetti, L., 1998). Creational design patterns are listed at the Table 2.1.

Table 2.1: Creational Design Patterns

Creational Patterns	**Description**	**Frequency of Use**
Abstract Factory	Creates an instance of several families of classes	High
Builder	Separates object construction from its representation	Medium Low
Factory	Creates an instance of several derived classes	High
Prototype	A fully initialized instance to be copied or cloned	Medium
Singleton	A class of which only a single instance can exist	Medium High

Source : http://www.dofactory.com/net/design-patterns

Structural Patterns: Purpose of Structural Patterns is to combine classes and objects, and to ensure that wider software groups are established. Structural design patterns are listed at the Table 2.2.

Table 2.2: Structural Design Patterns

Structural Patterns	**Description**	**Frequency of Use**
Adapter	Match interfaces of different classes	Medium High
Bridge	Separates an object's interface from its implementation	Medium
Composite	A tree structure of simple and composite objects	Medium High
Decorator	Add responsibilities to objects dynamically	Medium
Facade	A single class that represents an entire subsystem	High
Flyweight	A fine-grained instance used for efficient sharing	Low
Proxy	An object representing another object	Medium High

Source: http://www.dofactory.com/net/design-patterns

Behavioral Patterns: Behavioral Patterns are design patterns that define the common communication patterns among objects, and that realize such models. In that way, flexibility can be ensured at the stage of communication among objects. Behavioral design patterns are listed at the Table 2.3.

Table 2.3: Behavioral Design Patterns

Behavioral Patterns	**Description**	**Frequency of Use**
Chain of Resp.	A way of passing a request between a chain of objects	Medium Low
Command	Encapsulate a command request as an object	Medium High
Interpreter	A way to include language elements in a program	Low
Iterator	Sequentially access the elements of a collection	High
Mediator	Defines simplified communication between classes	Medium Low
Memento	Capture and restore an object's internal state	Low
Observer	A way of notifying change to a number of classes	High
State	Alter an object's behavior when its state changes	Medium
Strategy	Encapsulates an algorithm inside a class	Medium High
Template Method	Defer the exact steps of an algorithm to a subclass	Medium
Visitor	Defines a new operation to a class without change	Low

Source: http://www.dofactory.com/net/design-patterns

Thanks to design patterns, quality of applications increases. As the code will attain a standard structure, its performance will be increased and valuable time will be saved in solving a problem that has already been solved, as understandability of the code will also increase.

3. FRAMEWORK

3.1 WHAT IS A FRAMEWORK?

Framework are structures where sets of pre-built modules and libraries that are needed in projects can be found. New modules can be added to these modules as necessary.

The most important goal of frameworks is to save developers valuable time by avoiding to redesign the infrastructure of each project in their software development process; to concentrate on business processes, or, in short, to not have to re-write the code already written. In that way, development process is accelerated drastically as development costs decrease.

It is extremely important to develop and use frameworks in large-scale software projects. Large-scale projects will definitely require new developments or arrangements if the project does not fail. The use of frameworks will provide great advantages if the framework is designed to be flexible and modular.

Frameworks should have standards. Those who will use or develop further on the framework should comply with these standards. For a stable development life cycle, the framework must encourage developers to comply with these standards.

Developers might need to go through a training process in order to learn such rules, constrained by the framework to make code better quality. This process is a cost item within the software development process. For a framework to be used effectively, this cost is quickly overrun and turned into a profitable solution. In addition, this cost applies only to a project with an initial learning period. Once the learning process is completed, there will be no costs in other projects, and project-specific coding will be immediately available. As habits on using the frameworks develop, time required for project development will shorten.

If the framework is not to be developed but to be outsourced from another company, the purchase cost should be considered in addition to training costs.

Table 3.1: Framework cost table

Internal framework development:	**Cost = Duration of analysis + Development time + teaching / learning time**
Outsourced framework:	**Cost = Duration of analysis + Development time + Development fee + Learning time**
Purchasing a ready-made framework:	**Cost = Analysis time + Research time + Framework purchase fee + learning time.**

These costs for the frameworks that are listed at the Table 3.1, will be spent only for once. After the development and learning processes are completed, they can be used in other projects without any extra cost. There will definitely be additional costs to debug or fix errors in the framework along with new requests.

In addition, these costs will be added to the framework implementation phase for older projects that had been coded before the framework was developed, considerable for unfinished projects or framework infrastructures.

Omitting to use any frameworks could be the better choice if the framework is not supposed to be used very often or if the project to be developed is a small one, and if none or few of the modules provided by the framework are to be used.

The most effective frameworks for companies consist of modularized and combined existing common codes.

According to Wolgang Pree's article titled "Meta Patterns - A Means for Capturing the Essentials of Reusable Object-Oriented Design", frameworks must have hot spots and frozen spots. Frozen spots specify the general architecture of the framework. Components and the relation between components relate to this. There should not be frequent or sharp changes in these structures. The hot spots are where developers who use the framework will add their own code in line with the requirements of their projects (Pree W., 1994).

Frameworks provide high productivity and faster production thanks to their re-usability. In order to utilize from this advantage, the framework should be well-designed, and the applications that the company uses or to use should be analyzed correctly by developers. Experiences of individuals who previously developed an application for the company could be used for coding and analysis. Developers using the framework will need to understand the framework and its operation.

3.2 ADVANTAGES OF USING FRAMEWORKS

Frameworks contain the most commonly used modules and libraries in their structure. Therefore, the use of a framework speeds up the project during the development phase. There is no need to redevelop a previously developed module. These modules are either included within the framework or can be included and used very easily. Thanks to this characteristic, the project development process which would normally take up to 2 months could be completed in 2 to 3 weeks. Developers can begin developing in line with the requirements of the project on the first day. There will be no need to spend further time on modules that need to be used in most projects such as authorization, database connectivity, security, etc. as these are already provided by the framework.

Another advantage of using a framework is that new developers who participate in the project or are newly recruited by the company can be more easily integrated into development rules through the framework. Since such rules will already be defined by the framework, they will not go beyond these rules.

By virtue of frameworks, projects developed by different people can be consistent in style and standards of coding. This is crucial for achieving a flexible, extensible project that is as free as possible from errors. In addition, company employees who have not worked on that project before will be able to read the code easily. This allows a developer to rapidly integrate and work on a project that is not supervised by another developer or admin due to reasons such as annual leave, resignation, etc. This increases efficiency within the company at a very significant level.

With a well-designed framework, high-quality project outputs can be achieved even with inexperienced software developers.

3.3 RELATED WORKS

There are various works on Framework design and Development.

"Building Object-Oriented Frameworks" from Taligent Company, which produced Object Oriented Operating System as well as first operated with Apple and then with IBM, includes information about how a successful Framework design will be and information about how a Framework study will be productized (Building Object-Oriented Frameworks, Taligent, Inc.).

In the article "Building Object-Oriented Frameworks" written by Nik Boyd, according to trials in SmallTalk platform, experiments on the role of Frameworks play in system design and how the Framework should be built for the communication between the objects carried out. (Boyd N., 1993).

In the article "A framework for Object Oriented Frameworks Design", how the development of Framework design should be through user roles and components are explained. (Parsons, D., Rashid, A., Speck, A., & Telea, A., 1999).

In the article “Design, implementation and evolution of object- oriented frameworks: concepts and guidelines”, some issues such as multiple Framework studies and keeping up to date of Framework are mentioned and solutions to the problems are discussed. (Van Gurp, J., & Bosch, J. 2001)

In the article “Object-Oriented Application Frameworks”, Classification of Framework varieties has been made and strengths and weaknesses of Frameworks have been addressed as well as possible future trends with Framework have been mentioned. (Fayad, M., & Schmidt, D. C., 1997).

In the article “Object oriented framework development”, importance and benefits of Framework development have been explained, Information on how the structure of Framework should be has been given. Some potential possible problems have been studied and examined the Junit Framework as an example. (Markiewicz, M. E., & de Lucena, C. J., 2001).

3.4 ANTICIPATED MODULES IN CUSTOMIZED FRAMEWORKS

In this age, there are various types of frameworks. It is possible to find dozens of frameworks specific to any software language. These frameworks contain modules that make it easier to develop software on their own (.Net Framework, Spring, Zend etc.).

In this thesis, frameworks that companies produce to develop their own operations or structures are examined. In effect, the following titles cover frameworks of this type.

3.4.1 Authorization, Authentication and Security

In February 2017, Kaspersky Lab's chief cyber security and anti-virus software provider Eugene Kaspersky shared that the yearly loss caused by cybercrime to the world economy is between $400-500 billion.

Today, companies invest heavily to keep cybersecurity measures at the highest possible level due to possible loss of reputation, data, and concerns of other material losses.

One of the most important details of cybersecurity issues is software security. At this point, the framework software will need to accommodate necessary modules.

In framework software, security process starts at the authentication phase. The user must be able to login to the system once their identity is authenticated. In this process, usually user name or e-mail along with a password information will be required.

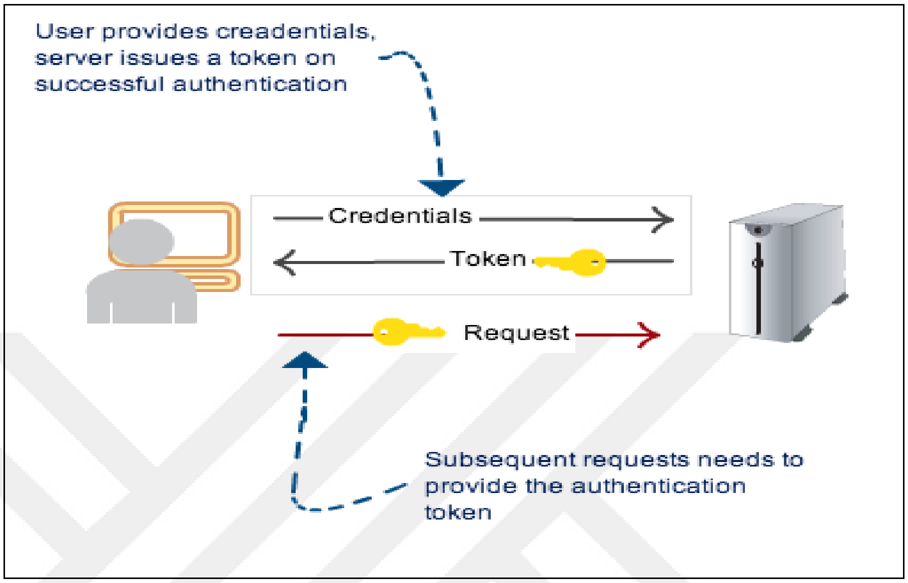
The issue of authentication is a broad topic and it can be the entire subject of another thesis, however, the following are some of the fundamental aspects that need to be considered:

- Password information should not be stored in plain text in the database. The ideal way is to pass it through a strong hash mechanism and retain. Hashing algorithms work in a single direction. Encrypted data cannot be decrypted. But nowadays some algorithms can be estimated with various methods. The most effective measure for this is to insert additional data to the data that will be hashed and put in the algorithm. This method is referred to as salt.

- Passwords must be difficult to predict, they should include a lengthy combination of lowercase and capital letters as well as special characters.

- Specific to web services, token systems with information should be used instead of requesting user name and password information each and every time. In token authentication, a token information is first obtained by entering user information. Following this process, this token information is added to each service request during the expiration period and the system receives the user ID with this token information (Figure 3.1).

Figure 3.1: Token Based Authentication Life Cycle



Source : http://www.gokhan-gokalp.com/asp-net-web-api-token-based-authentication/

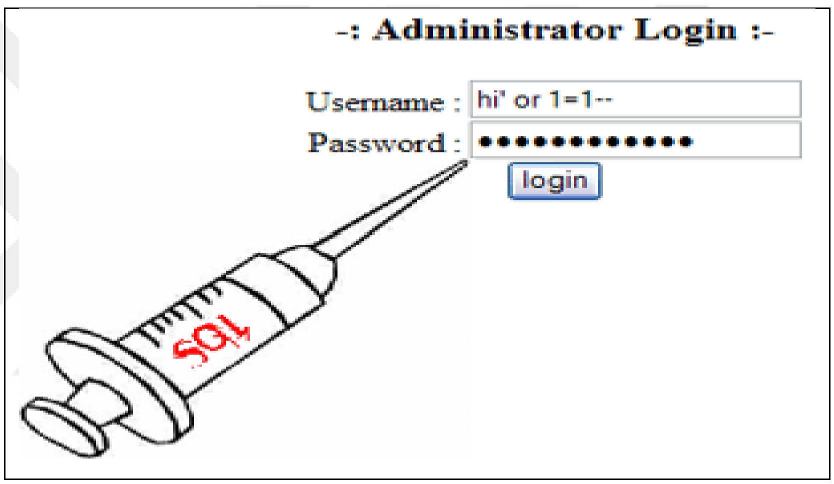
After the authentication stage, authorization stage follows. At this stage, access rights of the user are checked and determined in accordance with their identity. In other words, it is questioned here whether the authenticated user has permission to access the source they wish to access (Özbilgin, İ. G., & Özlü M., 2010).

These two issues can be managed in a similar way for each company. Consequently, it can be easily incorporated into the framework to be developed. For authentication and authorization stages that will be managed differently than other projects, these two properties should not be used in the framework but should be solved by coding specifically for the project.

In addition to authentication and authorization, certain other security measures must be taken. Database connection and database operations can be given as examples. If security measures are not in place during such transactions, the system may become vulnerable to SQL Injection attacks.

An SQL injection attack is the deviation of SQL queries operating in the system from their regular operation status by manipulation of the SQL statements by users intentionally or unintentionally. It is usually performed using meta characters such as quotes (“). Malicious users add additional queries to manipulate existing queries following the quotation mark (Halfond, W. G., & Orso, A., 2005).

Figure 3.2: SQL Injection Attack



Source : http://www.burakavci.com.tr/2014/01/sql-injection-login-bypass.html

An SQL injection attack is displayed in the Figure 3.2. After the attacker enters a string expression in the user name section, they end the string expression with a quote. Afterwards, if there are no measures in place within the system, it can be manipulated using SQL keywords.

*SELECT * FROM users WHERE username = “ + userName + “ AND password = “ + password*

These SQL statements work as follows after SQL injection attacks.

*SELECT * FROM users WHERE username = “hi“ OR 1 = 1 AND password = “ + password +” OR 1=1*

As a result of executing the above sentence, if there is a record in the database, no matter what the user name and password are, the record will definitely be returned. In effect, the system is manipulated (Boyd, S. W., & Keromytis, A. D., 2004).

These attacks are mitigated by the use of ORM tools within software at present. Necessary measures can be taken against these attacks via ORM tools. These ORM tools can also be used in frameworks that companies utilize.

Taking measures on the side of the framework which is a centralized system against SQL injections and other potential exploits will also protect other projects against such attacks without the necessity of any extra development.

3.4.2 Logging Transactions

Applications may not always work as expected. Any unexpected error or performance problem should be recorded in the application logs in order to track user activities.

3.4.3 FTP Transactions

File transfer among corporate computers as well as internal and external servers is a substantially common process. FTP protocol is needed in order to perform such operations. FTP protocol is a protocol where security is important, just like database operations. It should be preferred to operate under a framework, since both security concerns and codes that operate FTP transactions are usually standard.

3.4.4 Database Transactions

Databases are software that store data. The most important characteristic of databases is the ability to perform data operations such as recording, updating, querying, and deleting in a rapid manner.

Whether it is private corporation or a public authority, data is very important for any organization. Therefore, it is imperative to attach importance for the security of software that communicates with the database, and to the security of servers where database software are run.
There are several best practices in programming languages to avoid security vulnerabilities during database operations. Documentation of the software language used for these best practices can be reviewed.

Today, there are ORM tools that conduct CRUD transactions and database communication. These tools increase the quality of code, standardize it and increase security to the higher level. Recently, the majority of commonly used software languages have ORM-related libraries.

Companies that develop frameworks can either write their own ORM tool or choose among the ORM tools available on the market. This will provide significant gains in terms of security and faster coding.

3.4.5 Web Service Util Library

Web services are applications that exchange data over the web or the network. As communication is established on web or local network via data formats such as XML or JSON of which standards and communication rules are determined, they provides data exchange between applications coded in different programming languages and run on different operation systems.

There are a variety of web services by the names RESTful, SOAP, GraphQL or RPC.

RESTful web services are based on REST architecture. Access to resources is provided via WEB URIs whereas HTTP methods are used for communication (Hamad, H. & Saad, M. & Abed R., 2010).

The most commonly used methods are the following:

a. HTTP POST for data generation
b. HTTP PUT for data update
c. HTTP DELETE for data deletion
d. HTTP GET for data pulls

RESTful web services support both XML and JSON data formats. JSON offers the possibility of lower sized data transmission compared to XML. Lower sized data communication is extremely important in communication with mobile platforms. This has greatly increased the popularity of RESTful services in recent years.

SOAP is a document-based web service type and it supports XML format. Its advantage over RESTful services is that it contains a documentation characterized as WSDL. Through this documentation, rules for data communication between the server and the client are determined (Li, L., & Horrocks, I., 2004).

Although there are several libraries for documentation in RESTful services, there is no standardized use as in WSDL.

GraphQL has been on the market since 2015 and it is a kind of web service built on the principle of taking the strongest aspects of SOAP and RESTful services. The most obvious difference is that the requested data format is determined by the client (Vázquez-Ingelmo, A., Cruz-Benito, J., & García-Peñalvo, F. J., 2017).

It is crucial to determine rules of web services to be used in company software, to create abstractions that will force them to comply with these rules on the framework, and to aggregate modules to be shared into the framework in terms of inter-project code adaptation, standardization, security, and performance.

3.4.6 Other Modules

Depending on the industry a company operates in, module use may vary. Identifying common modules that companies use in their projects and adding these modules to their frameworks will provide significant advantages in terms of code quality, security, and time allocated to project coding.

4. FRAMEWORK CASE STUDY

Developed on Microsoft .Net Core platform, the source code of this framework case study can be accessed via the link below.
https://github.com/ykirkanahtar/CustomFramework

Microsoft .Net Core

It is the application development platform named .Net Framework, used by Microsoft since 2002. It is possible to develop web, desktop or console applications in C# or Visual Basic programming languages thanks to this platform. However, with changing and developing methodologies of software development, .Net Framework has lately been falling behind the expectations.

On grounds of both Net Framework’s slowpoke nature over the years and having fallen behind today’s expectations in software methodologies, Microsoft released .Net Core platform in May 2016.

In its version 2.1 preview2 as of April 2018, .Net Core platform has a more lightweight design compared to the classical .Net platform. Developers are able to use only the packages they need thanks to the platform’s modular structure.

It supports one of the most important software development approaches of today, the Dependency Injection processes. The platform even has its own basic Dependency Injection module, already built-in. Yet, the crucial distinctness of .Net Core is that the platform is open-sourced and independent.

To date, applications of .Net used to only run in Windows operating system without the support of certain 3rd party applications, but with .Net Core platform they are able to run in Windows, Linux and MacOS environments.

By courtesy of being an open-source platform, developers are able to contribute towards its development and expansion.

The modules below were utilized in this case study:

a) Database module (CustomFramework.Data)
b) Basic authorization module (CustomFramework.Authorization)
c) A module containing functions that can be used in all projects including web, console or services (CustomFramework.Utils)
d) WebApi template (CustomFramework.WebApiUtils)
e) Authorization module compatible with WebApi template (CustomFramework.WebApiUtils.Authorization)
f) A sample web-api project (CustomFramework.SampleWebApi)
g) Code generation module compatible with sample web-api project (CustomFramework.WebApiCodeGenerator)
h) E-mail module (CustomFramework.Email)
i) FTP module (CustomFramework.Ftp)

Unit tests have not been applied to this case study, but they should be in each projects of the companies.

4.1 DATABASE MODULE (CUSTOMFRAMEWORK.DATA)

In this framework case study, the responsibility of database communication was given to the project/module named CustomFramework.Data. Assuming this case study is a corporate project, all projects that'll utilize the internal database can perform their processes by use of this module.

Standard fields required as part of all projects are designated in database tables and entities, and thanks to this module, such fields were created in all tables. It will be enough to generate entities in the top modules from BaseModel class in order to utilize from this feature. In this case study, such fields were designated as follows: (Figure 4.1)

a) Id: Primary Key
b) CreateDateTime: Field that indicates when the related log was created
c) UpdateDateTime: Field that indicates the time of updates on the table, if any
d) DeleteDateTime: Field that indicates the time of deletion of logs
e) Status: Enum type field that indicates status of log as Active or Deleted

Figure 4.1: BaseModel

```
public class BaseModel<TKey> : IBaseModel<TKey>
{
    public TKey Id { get; set; }
    public DateTime CreateDateTime { get; set; }
    public DateTime? UpdateDateTime { get; set; }
    public DateTime? DeleteDateTime { get; set; }
    public Status Status { get; set; }
}
```

Use of this module enables the possibility to see when any log in any project was created without the need for any initiative from the developer responsible of the project. This, in fact, adds integrity to all projects within any company.

In some projects, information on the user who adds a log might be required. However, considering the fact that not all projects might have user info, such fields were not added to this shared module. These fields can be added to projects as necessary through top modules, or can be added to this module as an option.

Patterns and libraries used in the module are as follows:

EntityFramework Core: EntityFramework Core ORM tool was utilized for database communication. ORM tools constitute sort of a bridge between relational databases and object-oriented programming applications. They are used in creating relationships and communication between objects and databases.

Instead of statements such as INSERT, UPDATE, or SELECT; a library of .Net Framework named LINQ was utilized for database queries (Figure 4.2).

Figure 4.2: Linq Sample

```
var queryLondonCustomers3 =
                from cust in customers
                where cust.City == "London"
                orderby cust.Name ascending
                select cust;
```

Source: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/basic-linq-query-operations

Repository Pattern: Based on project needs, queries such as Create, Update, Delete, and Select can be used multiple times for an entity in various lines. The purpose of this pattern is to ensure re-usability of these queries.

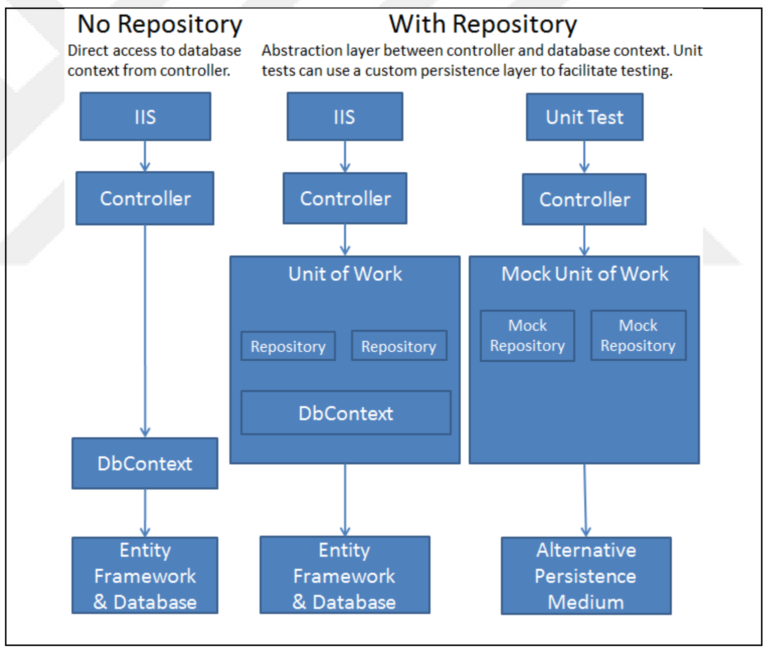
Queries are generally composed of long code blocks even in the cases using Linq. Usage of the same code block in various lines for the same query every time without using repository patterns is a code-smell leading to code repetition. Additionally, when the ingredients of a query is modified, the modification will have to be performed in each line. Consequently, critical errors arise when a single modification is skipped in any of the lines.

In repository patterns, there is a repository class for each entity, composed of queries regarding the entity. Top layers, generally referred to as business or service layers, call methods of this class when calling required queries. Top layers do not have to bother with what happens inside the method. Modifications on queries inside, accordingly, can be valid in each line that calls the method.

There are IRepository interface and BaseRepository class for the implementation of this pattern in this module. There are also methods of Add, Update, Delete, and Get to be utilized for all entities within this class.

Another advantage of repository pattern is the isolation of running ORM libraries from top layers. In that way, when the ORM tool is to be replaced in the future, top layers will not be affected by the replacement.

Figure 4.3: Repository Pattern with Unit of Work



Source: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

Unit of Work Pattern: Unit of Work Pattern performs all operations in the database under a single channel. This is significant for data integrity. If multiple objects in the database are to be updated simultaneously, undoing processes in cases of error is possible via this pattern.

The pattern can not only be used as standalone but also along with other repository patterns (Figure 4.3). For the use of UnitOfWork, there is a basic interface with class objects in the data module.

Npgsql: The data module in this case study supports two different types of databases such as MsSql and PostgreSql. Npgsql is an open-source library that provides fundamental classes for the communication between Microsoft .Net Framework and PostgreSql database.

In the sample project, "Npgsql.EntityFrameworkCore" library generated from "Npgsql" library was utilized to enable communication between PostgreSql and EntityFramework Core.

LinqKit: This library with an open-source code was used in creating dynamic linq queries. The operations were performed with PredicateBuilder class (Figure 4.4).

Figure 4.4: Use of LinqKit PredicateBuilder

```
IQueryable<Product> SearchProducts(params string[] keywords)
{
    var predicate = PredicateBuilder.New<Product>();

    foreach (string keyword in keywords)
    {
        string temp = keyword;
        predicate = predicate.Or(p => p.Description.Contains(temp));
    }
    return dataContext.Products.Where(predicate);
}
```

Source: https://github.com/scottksmith95/LINQKit#predicatebuilder

FluentApi: As EntityFramework models entities in accordance with database features, it provides a few alternative options to the developer. The most frequently used among the options is the method named FluentApi (Figure 4.6) in which DataAnnotations (Figure 4.5) are defined as Attributes within entity classes, coding the features in a separate class from entities.

Figure 4.5: Sample Use of Data Annotation

```
[Table("Orders")]
public class Order
{
    [Key]
    [Column("Id")]
    public int Id { get; set; }

    [Required]
    public DateTime DateTime { get; set; }

    [Required]
    [MaxLength(20)]
    public string Item { get; set; }

    [Required]
    public int Quantity { get; set; }

    [Required]
    [MaxLength(50)]
    public string Name { get; set; }

    [NotMapped]
    public string Note { get; set; }
}
```

Figure 4.6: Sample Use of Fluent API

```
public class OrderModelConfiguration<TEntity> : IEntityTypeConfiguration<TEntity>
        where TEntity : Order
{
    public void Configure(EntityTypeBuilder<TEntity> builder)
    {
        builder.ToTable("Orders");
        builder.Property(p => p.Id).ValueGeneratedOnAdd();
        builder.Property(p => p.DateTime).IsRequired();
        builder.Property(p => p.Item).HasMaxLength(20).IsRequired();
        builder.Property(p => p.Quantity).IsRequired();
        builder.Property(p => p.Name).HasMaxLength(50).IsRequired();

        builder.HasKey(p => p.Id);
    }
}
```

In addition to the libraries and functions mentioned above, the data module additionally consists of the features below:

a) To enable paging top modules based on the results from queries, a class that rotates the number of total logs on the query
b) Status Enum, holding log status
c) Extension classes consisting of ServiceCollections required in the use of this module in Asp.Net Core WebApi projects
d) Extension classes consisting of ServiceCollections for the choice of database, MsSql or PostgreSql
e) A class consisting of the function hashing in the use of fields to be kept and hashed, such as user password
f) SnakeCaseConverter class
    a. *(Note: It is recommended to use snake_case for PostgreSql, or PascalCase naming method for MsSql. As a matter of fact, C# entries are defined as PascalCase, and there is no need for additional adjustments in MsSql. As for the choice of PostgreSql, there is a SnakeCaseConverter class within the module.)*
g) EntityFramework library supports inserting in database the data defined as default when the application runs for the first time. (Like creating the first user) This process is called "Data Seeding". There are basic classes in the module for top modules to perform the Data Seeding process.

4.2 BASIC AUTHORIZATION MODULE (CUSTOMFRAMEWORK.AUTHORIZATION)

There is a basic authorization feature in this module to be used in a web-api project. This feature is utilized in the web-api project in adding attributes to Controllers. This attribute, named "permission", can be used in two ways according to the scenario in the framework case study. (Various scenarios may occur based on various corporate needs)

a) Entity Authorization

b) Customized Authorization

In Entity Authorization, the entity that owns Controller is also delivered to Permission Attribute as a string statement. An enum variable is also delivered to this attribute, where any of Create, Update, Delete or Select is chosen along with the entity string (as in Create User authorization).

Whereas in Customized Authorization, features specific to the project and independent of entity are delivered to Permission Attribute (as in UpdateUserEmail authorization).

The duty of this module is only to define boundaries for an Attribute. Responsibilities (authentication, etc.) beyond this point are at the discrete of the developer using this module. In the Figure 4.7, there is “PermissionAttribute” Code Block and the usage shown in the Figure 4.8.

Figure 4.7: Permission Attribute Code

```
    [AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple =
true)]
    public class PermissionAttribute : AuthorizeAttribute
    {
        public string Entity { get; set; }

        public string CustomClaim { get; set; }

        public Crud? Crud { get; set; }

        public PermissionAttribute(string customClaim) : base("Permission")
        {
            CustomClaim = customClaim;
        }

        public PermissionAttribute(string entity, Crud crud) : base("Permission")
        {
            Entity = entity;
            Crud = crud;
        }
    }
```

Figure 4.8: Sample Use of Permission Attribute

```
[Route("create")]
[HttpPost]
[Permission(nameof(User), Crud.Create)]
public async Task<IActionResult> Create([FromBody] UserRequest request)
{
    //....
}
```

Note: Another module that carries out authorization and authentication entirely was developed in his framework case study, as introduced in the pages following. The module to be introduced utilizes from “Basic Authorization Module” (CustomFramework.Authorization) in basic operations.

4.3 SHARED MODULE USABLE IN ALL PROJECTS (CUSTOMFRAMEWORK.UTILS)

Closely similar to convert methods, functions and modules required in all applications such as web, service or console are defined in this module.

Within the sample project, ToSnakeCase function (Figure 4.9), also used in Database module given in chapter 4.1, is defined in this module.

Figure 4.9: Convert Class

```
public static class Convert
{
    public static string ToSnakeCase(this string input)
    {
        if (string.IsNullOrEmpty(input)) { return input; }

        var startUnderscores = Regex.Match(input, @"^_+");
        return
            startUnderscores + Regex.Replace(input, @"([a-z0-9])([A-Z])", "$1_$2")
           .ToLower().Replace('ı', 'i');
     }
}
```

4.4 WEB API TEMPLATE (CUSTOMFRAMEWORK.WEBAPIUTILS)

Corporate firms quite often need communication between applications both internally and b2b. Web services with independent software languages are the most effective solution and communication platforms.

As in every module, web service applications also need to follow a standard for the benefit of the relation company. In that way, all web service projects will seem like they were coded by one single developer even if they were developed multiple developers. In effect, it enables developers outside the original team to easily perform operations such as maintenance, debugging, or adding in new calls.

This module was developed to set a standard for the sample project. In CustomFramework.Data module, repository classes were mentioned. While in this module, there are basic classes for use in the business layer.

The business layer is the layer where data from Controllers are interpreted and logged into the database after multiple validations. If there are database queries in the validations, these queries are not performed directly at this layer, but through a repository. Likewise, logging into the database is also performed via repositories and UnitOfWork object.

Features found in this module are as follows:

a) There is an ApiResponse class in the module in order to constitute a standard for objects returning from WebApi. In that way, the same format of information is provided from all webapis within the company, unless requested otherwise. This enables the client application to be easily integrated with all webapis when it needs to communicate with more than one.
b) It provides an interface named ILocalizationService for multilingual support
c) It undertakes bug management responsibility

d) Almost all entities have GetById methods for Create, Delete, and ID-based queries. Not necessarily for all of them, but the update method is also used for most entities. In effect, the module contains a BaseController class (Figure 4.10) with Create, Delete, and GetById methods as well as a BaseControllerWithUpdate class generated from BaseController for entities that'll used the Update method. In the event that top modules use these classes, they will have these methods by writing less code.
(Note: These classes can also be found in the next chapter's Web Api Authorization (CustomFramework.WebApiUtils.Authorization) module. There is no authorization in classes in this module, whereas authorization is utilized in Web Api Authorization module. Based on requirements, developers can implement classes in the related module)

Figure 4.10: BaseController Class

```
    public abstract class BaseController<TEntity, TCreateRequest, TResponse,
TManager, TKey> : Controller
        where TEntity : BaseModel<TKey>
        where TManager : IBusinessManager<TEntity, TCreateRequest, TKey>
    {
        protected readonly TManager Manager;
        protected readonly ILocalizationService LocalizationService;
        protected readonly ILogger<Controller> Logger;
        protected readonly IMapper Mapper;

        protected BaseController(TManager manager, ILocalizationService
localizationService, ILogger<Controller> logger, IMapper mapper)
        {
            Manager = manager;
            LocalizationService = localizationService;
            Logger = logger;
            Mapper = mapper;
        }

        [Route("create")]
        [HttpPost]
        public async Task<IActionResult> Create([FromBody] TCreateRequest
request)
        {
            var result = await Manager.CreateAsync(request);
            return Ok(new ApiResponse(LocalizationService,
Logger).Ok(Mapper.Map<TEntity, TResponse>(result)));
        }

        [Route("delete/{id:int}")]
        [HttpDelete]
        public async Task<IActionResult> Delete(TKey id)
        {
            await Manager.DeleteAsync(id);
            return Ok(new ApiResponse(LocalizationService, Logger).Ok(true));
        }

        [Route("get/id/{id:int}")]
        [HttpGet]
        public async Task<IActionResult> GetById(TKey id)
        {
            var result = await Manager.GetByIdAsync(id);
            return Ok(new ApiResponse(LocalizationService,
Logger).Ok(Mapper.Map<TEntity, TResponse>(result)));
        }
    }
```

4.5 WEB API UTIL WITH AUTHORIZATION AND AUTHENTICATION (CUSTOMFRAMEWORK.WEBAPIUTILS.AUTHORIZATION)

The previous chapter numbered 4.2 tackles CustomFramework.Authorization module. This is the module used in end-to-end authorization and user validation processes.

Being responsible for all end-to-end phases, it is found in most classes required in any web-api. User authentication process is performed via Tokens. Token Controllers are used for this purpose. ClientApplication info are requested in Login object for the validation of both User credentials and various devices or interfaces in the framework case study scenario. After successful authorization, a Token is generated via JwtManager class found in Authorization module and the info is delivered to the client along with an Expiry (Figure 4.11).

Figure 4.11: Login Method in TokenController Class

```
        [AllowAnonymous]
        [HttpPost]
        public async Task<IActionResult> Login([FromBody] Login login)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(ModelState.ModelStateToString(_localizationService));

            var clientApplication =
                await
_clientApplicationManager.LoginAsync(login.ClientApplicationCode,
login.ClientApplicationPassword);

            var user = await _userManager.LoginAsync(login.UserName,
login.UserPassword);

            var apiRequest = new ApiRequest(user, clientApplication);

            var claims = new List<Claim>
            {
                new Claim(typeof(IApiRequest).Name,
JsonConvert.SerializeObject(apiRequest)),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
            };

            var key = _token.Key;
            var issuer = _token.Issuer;
            var audience = _token.Audience;
            var expireInMinutes = _token.ExpireInMinutes;
            var token = JwtManager.GenerateToken(claims, key, issuer, audience,
out var expireDateTime, expireInMinutes);

            var tokenResponse = new TokenResponse
            {
                Token = token,
                ExpireInMinutes = expireInMinutes,
                RequestUtcDateTime = DateTime.UtcNow,
                ExpireUtcDateTime = expireDateTime,
            };

            return Ok(new ApiResponse(_localizationService,
_logger).Ok(tokenResponse));
        }
```

As for authorization process, models and their ModelConfiguration, Business, Validation, Request, Response, and Controller classes are found within the module.

a) **Role**: Used for user groups
b) **User**: Used for user operations
c) **UserRole**: Holds information on which user has which role
d) **Claim**: Holds customized authorization info
e) **RoleClaim**: Holds information on which roles are authorized to which claims
f) **UserClaim**: Holds information on which users are authorized to which claims
g) **RoleEntityClaim**: Holds information on which roles are authorized based on related entity.
h) **UserEntityClaim**: Holds information on which users are authorized based on related entity.
i) **ClientApplication**: Used in application authentication processes.
j) **ClientApplicationUtil**: Auxiliary model used for hashing the ClientApplicationPassword field
k) **UserUtil**: Auxiliary model used in hashing UserPassword field.

4.6 SAMPLE WEB API (CUSTOMFRAMEWORK.SAMPLEWEBAPI)

A basic webapi generated using Custom Framework module. Uses Data, WebApiUtils, WebApiUtilsAuthorization modules of CustomFramework.

4.7 CODE GENERATOR

Use of frameworks provides significant productivity by centrally gathering software rules in internal corporate projects as well as faster coding. However, in especially web-api projects, entity development phases specific to projects requires generation of lots of classes due to multi-layered architecture. (Repository, Business, Model, ModelConfiguration, Request, Response, Validation, Controller classes) Developing a code-generating software at this point will dramatically shrink development times.

Based on model fields and certain features (unique, primary key, foreign key, etc.) almost every class required by multi-layered architecture can be generated.
Use of Code Generators also provide the advantage of reducing error rates in addition to time. Especially in web-api projects containing excessive entities, developers might resort to copying and pasting related classes of previous entities to save time, leading to skipping to change one or more fields belonging to other entities inadvertently. This might as well mislead users during runtime. Such a possibility can be reduced to minimum by the use of Code Generator.

Yet, Code Generator might not be used for every scenario. Manual coding might also be required for certain nonstandard queries and methods.

In the case study, the code generator application was coded via Windows form project, a .Net Framework product. If required, code generation can also be possible via a Console application on .Net FrameworkCore platform so the platform will be independent.

4.8 FTP AND E-MAIL MODULE

There are also modules in the framework case study for FTP and E-Mailing processes required by almost all companies. These modules consist of one interface each and classes that implement the interfaces.

5. RESULTS

In this study, the importance of the use of framework is emphasized for projects of companies that develop software. Today, software technologies develop very rapidly and along with, issues such as security and quality of software projects become more and more important. The increase of investments made on software products and successful results as well as long-term nature of software projects make them a critical issue for companies in terms of not only their prestige but also financial gains.

Figure 5.1: Reasons for Project Cancellation

Reason for cancellation	Percentage of respondents (95% confidence interval)
Senior management not sufficiently involved	33 (13, 59)
Too many requirements and scope changes	33 (13, 59)
Lack of necessary management skills	28 (10, 54)
Over budget	28 (10, 54)
Lack of necessary technical skills	22 (6, 48)
No more need for the system to be developed	22 (6, 48)
Over schedule	17 (4, 41)
Technology too new; didn't work as expected	17 (4, 41)
Insufficient staff	11 (1, 35)
Critical quality problems with software	11 (1, 35)
End users not sufficiently involved	6 (0, 27)

Source: A Replicated Survey of IT Software Project Failures (El Emam, K. & Koru, A. G., 2008)

There is a scientific study in Figure 5.1 conducted in 2007, related to the reasons for software projects to result in failure. The use of the custom framework suggested in this thesis study can provide solutions to quality problems shown in Figure 5.1 in software projects, such as technical inadequacy, over schedule, or insufficient staff.

Advantages and gains of dividing software fragments (authentication, authorization, database communication, etc.) that are constantly used similarly or in the same way, and gathering them in a framework are investigated in this thesis study. Therefore, the advantages can be listed as follows:

a) Faster project delivery
b) Higher quality project output
c) More secure project output
d) Bringing coding standardization among projects
e) Better adaptable to modifications, flexible, and expandable project output
f) Easier integration among applications

The advantages can be analyzed item by item as below:

Faster project delivery: Instead of re-developing modules that are already coded and in use for other projects, using the already-developed one provides significant financial advantages for corporations and developers. Not having to invest their efforts on modules shared by multiple projects, developers can work on the flows specific to the project. This will, in effect, enable them to work with a better focus on the project.

Higher quality project output: Custom framework modules become more consistent, decisive, and higher quality products day by day as a result of experiences gained, errors occurred, and fixes performed for such errors throughout the period of their use. In this way, projects to use related modules will obtain the advantage of all the accumulated experience on the very first day.

More secure project output: Security in software projects is a critical element that needs umpteen amounts of effort. Negligence in such an issue can lead to significant financial losses for companies. By use of custom frameworks, as in "higher quality project output" mentioned above, security developments and updates from previous experiences will be transferred automatically to newly developed projects. In addition, fixing newly-detected security gaps within the custom framework will enable all projects using the same framework to benefit from the update.

Bringing coding standardization among projects: Every developer differs from another in their level of knowledge. Despite the fact that terms of employment are among the important criteria in job advertisements for candidate developers, the level of knowledge of an idealist developer of 1 to 2 years can become more than another developer with 4 to 5 years of experience, not bothering to refresh or increase their knowledge on recent developments. Differences between levels of knowledge among developers will be a determining factor having an impact on the project success as in quality, security, or deadline. As it will force developers to code under certain restrictions, use of CustomFramework will reduce to minimum the quality differences due to various levels of knowledge on developers' side.

Better adaptable to modifications, flexible, and expandable project output: Almost all software projects continue to live after being taken into production environment or delivered to customer. Be it new requests or bugs discovered in the future and further modifications, it will lead to the continuation of a delivered software project. This process of continuation does not mean failure for the software project. However, new requests and modifications in too many parts of the code will have a negative impact on its integrity, reducing the lifetime of the project. This, later, turns into a process leading into failure. In effect, maximum compliance with the rules mentioned in the 2nd chapter of this thesis is critical in terms of flexibility, expandability, and integrity of software project. Such rules were determined by significant authorities of the world of software, decades of experience, recommended in practice. The higher the compliance to these rules is in the custom framework products to be used by corporation (except for problems stemming from the customer or project management, etc.), the longer the life of the software product will be.

Easier integration among applications: Be it internal or inter-organizational, communication among applications will be required. A custom framework can also be used for the integrity of communication between internal applications. Unless required otherwise based on type of project, using the same Response format will dramatically increase the speed of development.

5.1 SAMPLE FRAMEWORK DEVELOPMENT COST AND GAIN OF USAGE

The use of measurement methods in software projects is critical to determine project costs, allocate accurate resources, and deliver timely. For measurements to be performed correctly, software size estimation methods will need to be used (Borandağ, E., Yücalar, F., & Şahinaslan, Ö).

Being easy, the use of Lines of Code (LOC) is the most common method. However, differentiating factors such as programming language and developer's knowledge have an impact on the measurement results.

Applying the LOC method on the framework given under chapter 4 of the thesis yields the results in Table 5.1.

Table 5.1: LOC Measurement for Sample Framework

Project Name	**Total Number of Lines**
CustomFramework.Data	639
CustomFramework.Authorization	177
CustomFramework.WebApiUtils	834
CustomFramework.WebApiUtils.Authorization	4162
CustomFramework.Email	69
CustomFramework.Ftp	395
Total	**6276**

According to these results, a software project to utilize from all 6 modules in Table 5.1 for a sample framework project assumed to be developed beforehand, 6276 lines of code will not require extra effort. (Based on project needs, integration or certain differences might slightly require extra effort. The assumption above considers projects that will utilize from projects one by one.)

Repetition codes can be found in projects where Custom Frameworks will be used. For instance, for each entity in a webapi project, there are Create/Update/Get controllers, business layers called by the controllers, and data layers behind the business layers. Ruling out entities and field names of entities, there will be little to no difference in the code flows. Implementing Code Generator applications in Custom Frameworks for such repetition codes is also recommended in this study.

The software project to use the modules in the framework case study is represented with the name SampleWebApi.

In SampleWebApi project, it is recommended to use the above-mentioned code generation tools for entities to be included in the project. When code generators are used, it will provide an advantage at the amount of development durations of lines of code given in Table 5.2.

Table 5.2: LOC Measurement Based on Entity

Entity	Lines of Code
Course	303
Student	301
Teacher	280
StudentCourse	247
Total Gain	**1132**

Note: As methods will be generated through code generation tool, which are standardized and usable in classes of each entity, necessary extra methods will require extra effort. This will vary depending on the type and function of the entity.

Functional Size Measurement methods were developed due to the measurement disadvantages posed by differences in programming language and developer's knowledge in the LOC method.

Function Points and Function Point Analysis (FPA) methods were first developed by Allan Albrecht as an alternative to lines of code.

In 1983, Functional Size Measurement (FSM) method was developed by Allan Albrecht and John Gaffney (Fetcke T., Abran A., & Dumke, R. R., 2016). In this method, functionality of the software forms the basis. A myriad of FSM methods was later developed with modifications to the FPA method by various parties.

a) Function Points - FP,
b) IFPUG Function Points Analysis – IFPUG FPA,
c) Mark II Function Points – MK II FP,
d) Nesma Function Points,
e) Full Function Points – FFP,
f) COSMIC Full Function Points – COSMIC FFP,
g) Object Points,
h) Object- Oriented Function Points – OO FP,
i) Object-Oriented Method Function Points – OOmFP

Function Points approach oversees productivity designated as person/month.
In this approach, if inputs and outputs can be estimated, a scoring calculation can be performed by using the parameters and results can be converted into LOC. Cost, effort, and

duration forecasts can be performed via the LOC. Unadjusted Function Point Calculation is presented in Table 5.3 based upon weight ratio (Borandağ, E., Yücalar, F., & Şahinaslan, Ö).

Table 5.3: Unadjusted Function Point (UFP) Calculation

	Low Complexity	Medium Complexity	High Complexity
External Inputs	n x 3	n x 4	n x 6
External Outputs	n x 4	n x 5	n x 7
External Inquiries	n x 3	n x 4	n x 6
Internal logical files	n x 7	n x 10	n x 15
External interface files	n x 5	n x 7	n x 10

Source: https://blog.chenchatech.com/tag/functional-points/

(n): Number of classes

Performing a cost estimate for the project before developing it, bearing in mind the framework case study, would enable the following steps to be performed. UFP Calculations for the Case Study's modules are listed at the Table 5.4 to Table 5.9

Table 5.4: UFP Calculation for WebUtils.Authorization in the Case Study

	Low Complexity	Medium Complexity	High Complexity
External Inputs	0 x 3	0 x 4	0 x 6
External Outputs	0 x 4	0 x 5	0 x 7
External Inquiries	28 x 3	3 x 4	0 x 6
Internal logical files	0 x 7	0 x 10	0 x 15
External interface files	0 x 5	0 x 7	0 x 10
UFP Point	**96**		

Table 5.5: UFP Calculation for Authorization Project in the Case Study

	Low Complexity	Medium Complexity	High Complexity
External Inputs	1 x 3	0 x 4	0 x 6
External Outputs	0 x 4	0 x 5	0 x 7
External Inquiries	0 x 3	0 x 4	0 x 6
Internal logical files	0 x 7	0 x 10	0 x 15
External interface files	0 x 5	0 x 7	0 x 10
UFP Point	**3**		

Table 5.6: UFP Calculation for Data Project in the Case Study

	Low Complexity	Medium Complexity	High Complexity
External Inputs	0 x 3	0 x 4	0 x 6
External Outputs	0 x 4	0 x 5	0 x 7
External Inquiries	0 x 3	0 x 4	0 x 6
Internal logical files	1 x 7	0 x 10	0 x 15
External interface files	0 x 5	0 x 7	0 x 10
UFP Point	**7**		

Table 5.7: UFP Calculation for WebApiUtils Project in the Case Study

	Low Complexity	Medium Complexity	High Complexity
External Inputs	0 x 3	0 x 4	0 x 6
External Outputs	5 x 4	0 x 5	0 x 7
External Inquiries	0 x 3	0 x 4	0 x 6
Internal logical files	0 x 7	0 x 10	0 x 15
External interface files	0 x 5	0 x 7	0 x 10
UFP Point	**20**		

Table 5.8: UFP Calculation for FTP Project in the Case Study

	Low Complexity	Medium Complexity	High Complexity
External Inputs	0 x 3	0 x 4	0 x 6
External Outputs	0 x 4	0 x 5	0 x 7
External Inquiries	0 x 3	0 x 4	0 x 6
Internal logical files	0 x 7	0 x 10	0 x 15
External interface files	1 x 5	0 x 7	0 x 10
UFP Point	**5**		

Table 5.9: UFP Calculation for E-Mail Project in the Case Study

	Low Complexity	Medium Complexity	High Complexity
External Inputs	0 x 3	0 x 4	0 x 6
External Outputs	0 x 4	0 x 5	0 x 7
External Inquiries	0 x 3	0 x 4	0 x 6
Internal logical files	0 x 7	0 x 10	0 x 15
External interface files	1 x 5	0 x 7	0 x 10
UFP Point	**5**		

Following Function Point calculation, a Value Adjustment Factor is calculated based upon responses to questions regarding general system characteristics, composed of 14 items on implementation difficulty. Questions are as presented in the Table 5.10.

Table 5.10: General System Characteristics

1) Data Communications	**8) Online update**
2) Distributed Data Processing	**9) Complex processing**
3) Performance	**10) Reusability**
4) Heavily used configuration	**11) Installation case**
5) Transaction rate	**12) Operational case**
6) Online data entry	**13) Multiple sites**
7) End-user efficiency	**14) Facilitate change**

Source: http://www.lnse.org/papers/46-IE2038.pdf

A value between 0 and 5 is given for each question above, and the values are added to calculate Degree of Influence (Raju, H. K., & Krishnegowda, Y. T., 2013).

0 – Not present or no influence

1 – Incidental influence

2 – Moderate influence

3 – Average influence

4 – Significant influence

5 – Strong influence throughout

$$VAF = (TDI \times 0.01) + 0.65$$

Accordingly, general system characteristics scores for the framework case study are as shown at the Table 5.11.

Table 5.11: General System Characteristics Scores for the Case Study

GSC Questions	Authorization	Data	WebApi Utils	WebApiUtils Authorization	FTP	EMail
1	0	5	0	0	2	2
2	0	5	0	0	3	3
3	0	5	3	2	4	4
4	0	5	1	1	2	2
5	3	5	3	3	2	2
6	0	5	0	0	3	3
7	0	0	5	5	4	4
8	0	0	0	0	5	0
9	0	1	2	2	0	0
10	5	5	5	5	5	5
11	5	5	5	5	5	5
12	0	0	0	0	0	0
13	5	5	5	5	5	5
14	5	5	5	5	5	5
TDI Score	**23**	**51**	**34**	**33**	**45**	**40**
VAF Score	**0.88**	**1.16**	**0.99**	**0.98**	**1.1**	**1.05**

After obtaining VAF score, the formula below is used to get Function Points.

$$FP = UFP \text{ X } VAF$$

Weight ratios differ from one programming language to another. In effect, to obtain LOC value after obtaining function point, Programming Language Statements per Function Point value is multiplied by FP value. The programming languages statements per function points are listed at the Table 5.12

Table 5.12: Programming Language Statements Per Function Point

Language	Minimum (Minus 1 Standard Deviation)	Mode (Most Common Value)	Maximum (Plus 1 Standard Deviation)
C	60	128	170
C#	40	55	80
C++	40	55	140
Cobol	65	107	150
Fortran 90	45	80	125
Fortran 95	30	71	100
Java	40	55	80
Macro Assembly	130	213	300
Perl	10	20	30
Second generation default (Fortran 77, Cobol, Pascal, etc.)	65	107	160
Smalltalk	10	20	40
SQL	7	13	15
Third generation default (Fortran 90, Ada 83, etc.)	45	80	125
Microsoft Visual Basic	15	32	41

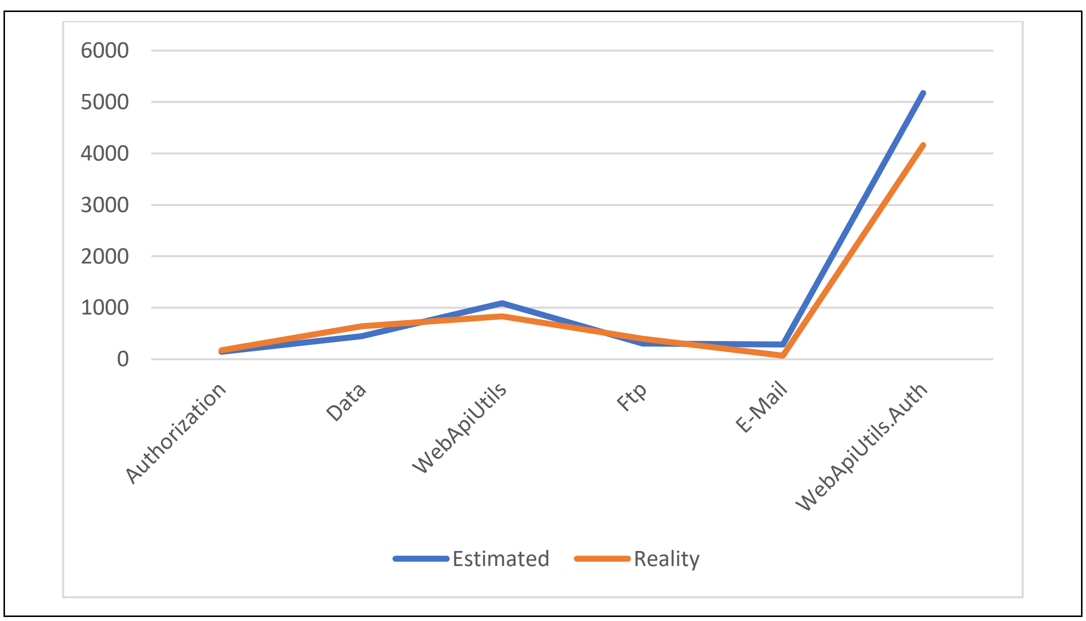
Source: Adapted from Estimating Software Costs (Jones 1998), Software Cost Estimation with Cocomo II (Boehm 2000), and Estimating Software Intensive Systems (Stutzke 2005).

Table 5.13: Calculated Lines of Code in the Framework Case Study

Projects	UFP Points	VAF Points	FP Points	Estimated LOC (KLOC) (C# => FP x 55)	LOC in Reality
Authorization	3	0.88	2.64	145.2	177
Data	7	1.16	8.12	446.6	639
WebApiUtils	20	0.99	19.8	1089	834
WebApiUtils. Authorization	96	0.98	94.08	5174.4	4162
Ftp	5	1.1	5.5	302.5	395
E-Mail	5	1.05	5.25	288.75	69

It was observed that the calculated LOC values in other projects except for E-mail project in Table 5.13 are similar to actual LOC values.

Figure 5.2: LOC Estimated and LOC in Reality



In light of the data, the formula below is used in the calculation of effort via the COCOMO method.

$$\text{Effort} = 2.4 \times (\text{KLOC})^{1.05} \ \text{Person/Month}$$

**(KLOC = LOC / 1000)*

For estimated time to spend, the formula below is used.

$$\text{Duration} = 2.5 \times (\text{Effort})^{0.38} \ (\text{Month})$$

Table 5.14: Development Cost

Projects	**(KLOC)**	**Effort (Person/Month)**	**Duration (Month)**
For Estimated Project	7.446	19.757	7.768
For Sample Project	6.276	16.511	7.255

Effort and duration of development mentioned in the Table 5.14 can be considered as much a gain for each new project as an initial cost, since in projects to use the ready-made modules no additional development cost will incur, or if modifications are necessary, much less coding effort will have to be made for integration.

For instance, when it is required to code from scratch all modules in a CustomFramework in a project that does not utilize from Custom Frameworks, the effort will take around 6,000 to 7,000 lines of code.

As a second example, assuming there are 30 projects at a company where database module is coded from scratch, an extra effort will take 19,170 lines of code (30 x 639) based on the number of lines of database module in the Case Study.

Furthermore, considering the authorization processes in 20 of these 30 projects, an additional effort for 83,240 lines of code (20 x 4,162) will be necessary based on the number of lines in the WebApiUtils.Authorization module in the CustomFramework. Time spent on other efforts such as debugging, maintenance, or modification will increase exponentially when added onto the abovementioned inconveniences. For example, it will be necessary to perform the same modification in every project one by one when there’s need for change of brand in the database product used by the management. In practice, this will result in large scale modifications on the code, necessitating similarly large-scale costs in addition to the extra time. In fact, depending on the number of projects, it might even be mandatory to waive from any desired modification.

In light of such potential scenarios, it can be put forward that the higher the number of projects at a company, the higher the advantages of utilizing from Custom Frameworks will be.

Finally, quality measurement of the framework case study has been made by using SonarQube 6.7.3 (https://www.sonarqube.org) and NDepend 2018.1.1 (https://www.ndepend.com) . Both of them are code analysis applications.

SonarQube reports by scanning bugs under the title of Reliability, vulnerability under the title of security, code smells under the title of maintainability, unit tests under the title of coverage as well as duplication codes, code complexity and application size for over 20 for software languages.
Results have been graded A, B, C, D and E in the reports of Reliability, Security and Maintainability respectively from good to bad.

Complexity of each class is being calculated in the Complexity report and complexity count between 1- 4 are evaluated as good, complexity count between 5-7 are evaluated as acceptable, complexity count between 8-10 are evaluated as reviewable and complexity count over 11 are evaluated as must be reviewed.

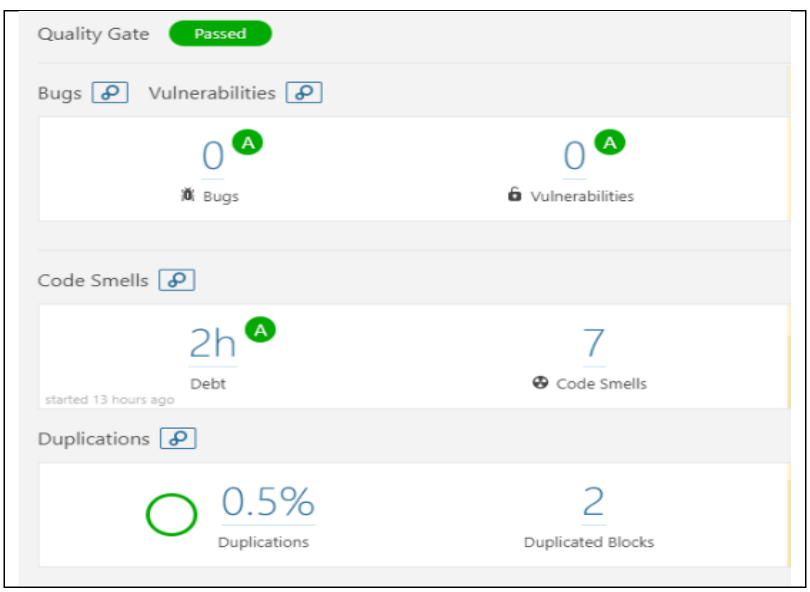
Duplicated lines are given and a percentage rate will be submitted for the whole project scope in the Duplication report. The projects having percentage rate over 3% should be reviewed.

The results at the Table 5.15 and Figure 5.3 are obtained when Case Study Project is analyzed through SonarQube.

Table 5.15: SonarQube Analysis Report for Framework Case Study

Reliability	A
Security	A
Maintainability	A (Count of Code smells: 7)
Duplications Density	% 0.5
Duplicated Lines Count	42
Number of classifications which have five and over five complexities	9 (Total class count : 262)

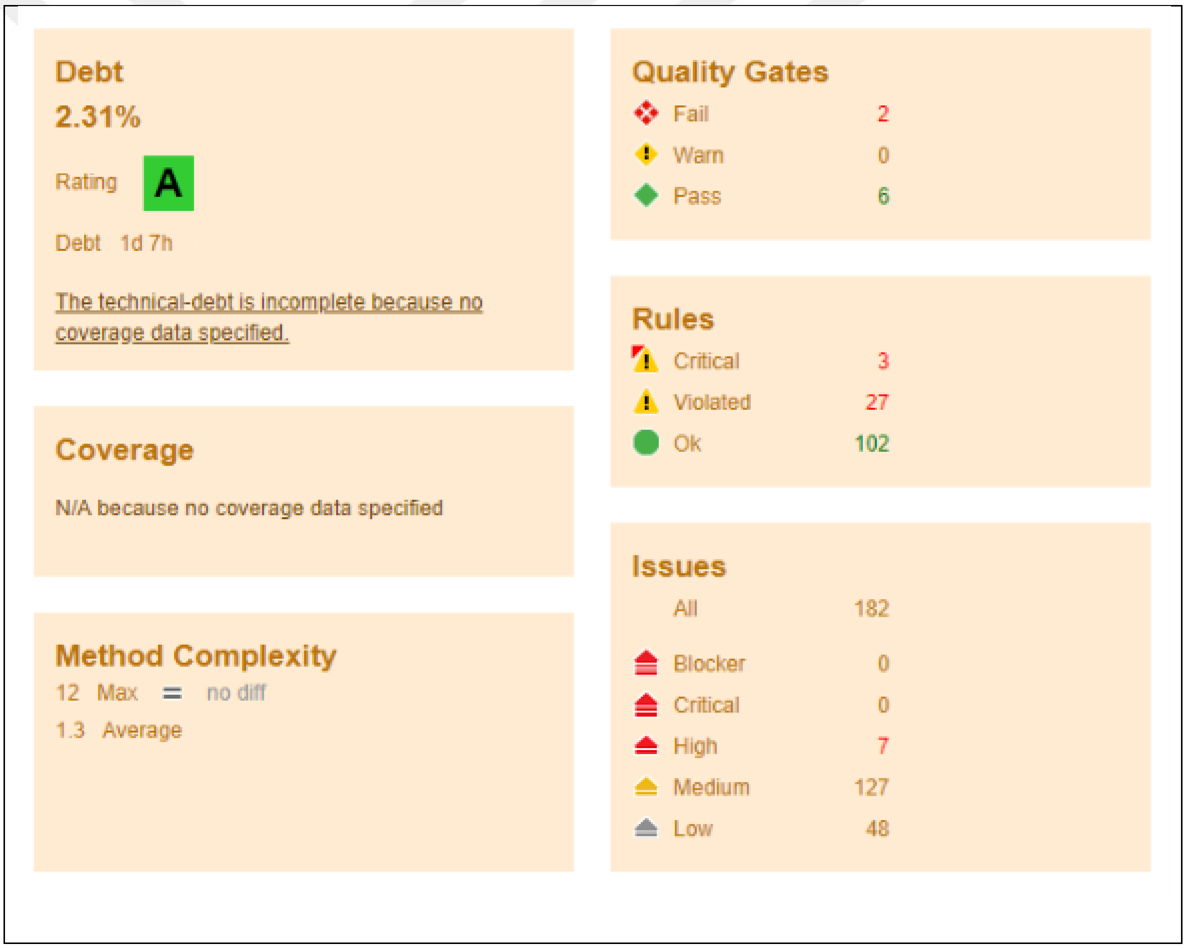
Figure 5.3: SonarQube Report For Framework Case Study



NDepend is also scans the source code and it generates interactive reports about .Net code quality and code rules compliance. After scanning Case Study sour code, NDepend gives also A Rating for it. The result is displayed in the Figure 5.4

The 42 of the 183 issues in this report is related to the non-static class that are not being instantiated but on the other hand, because those classes are initialized with Dependency Injection, these issues are false positive.

Figure 5.4: NDepend Report For Framework Case Study



6. CONCLUSION

Programming languages improve day by day and provide more functionalities with less coding required. For instance, three to four lines of code would be required for even opening or closing a database connection in the past, whereas now entering a query into the database can be performed in two to three lines from start to finish.

No matter how software technologies make things easier for developers, they still need to make effort in developing as well as abiding by certain rules. With the recent increase of distributed architecture, network connection technologies and the security of software using them have become more prominent.

Meanwhile, due to developing connection speeds, growing hardware resources, and increasing competition among corporations, companies now grow their applications to better serve their users with more functionalities. For instance, applications sending solely plain text messages in the past are now capable to provide video conversation as well as file, photo, and video exchange functionalities. In effect, there are more numbers of classes and entities in the back-end of applications, the ease of coding and adding new functionalities to which has become more important for the long-term success of applications.

Be it for security, error management, easier management of classes, or even entities arising from each newly-added functionality, developers tend to adapt to multi-layered architectural designs as well as certain recommended rules during coding.

In this study, the advantages and importance of using custom frameworks as well as the layers and rules mentioned above are emphasized in developing under certain restrictions set by corporations for corporate-specific requirements in addition to recommendations by global software authorities.

Thanks to custom frameworks, companies are able to not only save time on modules developed already but also develop faster, while at the same time reducing to minimum quality differences stemming from differences in levels of knowledge among developers as well as performing project maintenance and fixes in a much quicker and a more efficient manner.

It was observed that an unnecessary effort for approximately 7,000 lines of code can be avoided when all modules are used for products similar to the framework product described under the Framework Case Study chapter of the paper. The higher the number of projects at a company, the higher the savings on efforts will be; meanwhile thanks to a centralized Framework system, potential bugs and problems can be solved in a much more effective way and, in effect, Frameworks can contribute directly towards the success levels of projects within any company.

Today, information technologies grow and change especially quickly. Companies that are able to keep pace with the change survive, whereas most of those who cannot keep pace unfortunately either have grave troubles, go bankrupt, or even are sold to other companies.

The purpose of this thesis is to draw attention to the importance of the use of custom frameworks and to follow coding standards, decreasing ratios of failure in software technologies which are among the most important requirements as of today, contributing towards global economy, and ensuring more qualified, knowledgeable and meticulous developers to thrive in the market by reducing inexperienced developers' adaptation period.

REFERENCES

Books

Acar, Ö., 2012. *Java tasarım şablonları ve yazılım mimarileri.* Istanbul, Pusula Yayıncılık ve İletişim

Ferthers, M. C., 2004. *Working effectively with legacy code.* Upper Saddle River NJ, Prentice Hall PTR

Fowler, M. & Beck, K., 1999. *Refactoring: improving the design of existing code.* Boston, Addison-Wesley Longman Publishing Co., Inc

Gökalp, G. & Tungut, H.B., 2016 *Asp.Net Web API ve Mimari Özellikleri.* Istanbul, Kodlab Yayın Dağıtım Yazılım Ltd.

Martin, R. C., 2008. *Clean code a handbook of agile software craftsmanship.* Upper Saddle River NJ, Prentice Hall PTR

Osherove, R., 2009. *The art of unit testing with examples in C#.* Second Edition, Shelter Island NY, Manning

Seeman, M., 2011. *Dependenct injection in .net.* Shelter Island NY, Manning

Smith, S., 2017. *Architecting modern web applications with asp.net core and microsoft azure*. Redmond Washington, Microsoft Corporation

Taşdelen, A., 2016. *C++, Java ve C# ile UML ve Dizayn Paternleri.* Istanbul, Pusula Yayıncılık ve İletişim

Periodicals

Antoniol, G., Fiutem, R., & Cristoforetti, L., 1998. Using metrics to identify design patterns in object-oriented software. In *Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International IEEE*. pp. 23-24

Beydağlı, E., Kara, M., Bahşi, H., & Alparslan, E., 2010. Güvenli Yazılım Geliştirme Modelleri ve Ortak Kriterler Standardı. *Ulusal Yazılım Mühendisliği Sempozyumu*, 8.

Borandağ, E., Yücalar, F., & Şahinaslan, Ö. Yazılım Projelerinde Büyüklük Tahmini.

Boyd, N. 1993. Building object–oriented frameworks. *The Smalltalk report*, **3**(1), pp.1-16.

Boyd, S. W., & Keromytis, A. D., 2004. SQLrand: Preventing SQL injection attacks. In *International Conference on Applied Cryptography and Network Security*. Springer, Berlin, Heidelberg. pp. 292-302

Bulychev, P., & Minea, M., 2008. Duplicate code detection using anti-unification. In *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering (No. 2)*

Chebanyuk, E., & Markov, K., 2016. An approach to class diagrams verification according to SOLID design principles. In *Model-Driven Engineering and Software Development (MODELSWARD), 2016 4th International Conference on IEEE*. pp. 435-441

Coram, M., & Bohner, S., 2005. The impact of agile methods on software project management. In *Engineering of Computer-Based Systems, 2005. ECBS'05. 12th IEEE International Conference and Workshops on the IEEE*. pp. 363-370

El Emam, K. & Koru, A. G., 2008. A Replicated Survey of IT Software Project Failures. IEEE Software **25**(5)

Fayad, M., & Schmidt, D. C. (1997). Object-oriented application frameworks. *Communications of the ACM*, **40**(10), pp.32-38.

Fetcke T., Abran A., & Dumke, R. R., 2016. 2.1 A Generalized Representation for Selected Functional Size Measurement Methods. *COSMIC Function Points: Theory and Advanced Practices*, pp. 89.

Fontana, F. A., Dietrich, J., Walter, B., Yamashita, A., & Zanoni, M., 2016. Antipattern and code smell false positives: Preliminary conceptualization and classification. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on IEEE*, **1**, pp. 609-613

Halfond, W. G., & Orso, A., 2005. AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering ACM*. pp. 174-183

Hamad, H. & Saad, M. & Abed R., 2010. Performance Evaluation of RESTful Web Services for Mobile Devices. *International Arab Journal of e-Technology*, **1**, (3), pp. 72-78

Hunt, A., & Thomas, D., 2002. Software archaeology. *IEEE Software*, **19**(2), pp. 20-22.

Van Gurp, J., & Bosch, J. (2001). Design, implementation and evolution of object oriented frameworks: concepts and guidelines. *Software: Practice and Experience*, **31**(3), pp.277-300.

Kim, S., Whitehead Jr, E. J., & Zhang, Y., 2008. Classifying software changes: Clean or buggy?. *IEEE Transactions on Software Engineering*, **34**(2), pp. 181-196.

Li, L., & Horrocks, I., 2004. A software framework for matchmaking based on semantic web technology. *International Journal of Electronic Commerce*, **8**(4), pp. 39-60.

Liskov, B., 1988. Keynote address-data abstraction and hierarchy. *ACM Sigplan Notices*, **23**(5), pp. 17-34.

Mackinnon, T., Freeman, S., & Craig, P., 2000. Endo-testing: unit testing with mock objects. *Extreme programming examined*, pp. 287-301.

Madasu, V. K., Venna, T. V. S. N., Eltaeib, T., Moalla, M. A., Almuslet, N. A., Alshake, A. A., ... & Soni, S. *SOLID Principles in Software Architecture and Introduction to RESM Concept in OOP. Studies*, **35**(38), pp. 8.

Markiewicz, M. E., & de Lucena, C. J. (2001). Object oriented framework development. *Crossroads*, **7**(4), pp.3-9.

Parsons, D., Rashid, A., Speck, A., & Telea, A. 1999. A" framework" for object oriented frameworks design. In Technology of Object-Oriented Languages and Systems, *1999. Proceedings of IEEE*. pp. 141-151.

Pree W., 1994. Meta patterns — A means for capturing the essentials of reusable object-oriented design. In *European Conference on Object-Oriented Programming*, pp. 150-162

Raju, H. K., & Krishnegowda, Y. T., 2013. Software sizing and productivity with Function Points. *Lecture Notes on Software Engineering*, **1**(2), pp. 204.

Razina, E., & Janzen, D. S., 2007. Effects of dependency injection on maintainability. In *Proceedings of the 11th IASTED International Conference on Software Engineering and Applications*: Cambridge, MA, p. 7

Rehber, D. Yazılım Projelerinde Başarısızlık.

Özbilgin, İ. G., & Özlü M., 2010. Yazılım Geliştirme Süreçleri ve ISO 27001 Bilgi Güvenliği Yönetim Sistemi.

Özçelik, O., & Altilar, D. T., 2015. Test-Driven Approach for Safety-Critical Software Development. *JSW*, **10**(7), pp. 904-911.

Singh, H., & Hassan, S. I., 2015. Effect of SOLID Design Principles on Quality of Software: An Empirical Assessment. *International Journal of Scientific and Engineering Research*, **6**(4).

Sjøberg, D. I., Yamashita, A., Anda, B. C., Mockus, A., & Dybå, T., 2013. Quantifying the effect of code smells on maintenance effort. *IEEE Transactions on Software Engineering*, **39**(8), pp. 1144-1156.

Suresh, P. & MuthuKumaran, S., 2013. Refactoring and Detection of Bad Smells of Coding Using Larger Scale and Critical Incident Technique. In *International Journal of Science and Research*, ISSN (Online): 2319-7064,

Tiftik, N., Öztarak, H., Ercek, G., & Özgün, S., 2007. Sistem/yazılım geliştirme sürecinde doğrulama faaliyetleri. *III. Ulusal Yazılım Mühendisliği Sempozyumu, Ankara*, pp. 1-2.

Ural, E., Tekin, U., & Buzluca, F., 2008. Nesneye dayalı yazılım metrikleri ve yazılım kalitesi. *Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu*.

Vázquez-Ingelmo, A., Cruz-Benito, J., & García-Peñalvo, F. J., 2017. Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL. In

Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality. ACM. pp. 89

Vlissides, J., Helm, R., Johnson, R., & Gamma, E., 1995. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, **49**(120), pp. 11.

Other Sources

Acar. Ö., 2014, Kokan Kod – 1. Bölüm, http://www.pratikprogramci.com/2014/11/15/kokan-kod-1-bolum/. [accessed 10.10.2017]

Behavioral Patterns, https://sourcemaking.com/design_patterns/behavioral_patterns. [accessed 05.11.2018]

Bloomberght, 2017. Siber suçların yıllık zararı 400-500 milyar dolar, http://www.bloomberght.com/haberler/haber/1983567-siber-suclarin-yillik-zarari-400-500-milyar-dolar. [accessed 02.12.2017]

Dorman, S., 2007, Why Coding Standards Are Important, http://scottdorman.github.io/2007/06/29/Why-Coding-Standards-Are-Important/. [accessed: 04.10.2017]

Fowler, M., 2006, Code Smell, https://martinfowler.com/bliki/CodeSmell.html. [accessed 10.10.2017]

Hastie, S., Wojewoda S, 2015. Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch, https://www.infoq.com/articles/standish-chaos-2015. [accessed: 04.10.2017]

Kurtel, K., Eren Ş., "Yazılım Ölçümü: Genel Bir Bakış," Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu (YKGS), 2008.

LinqKit, https://github.com/scottksmith95/LINQKit [accessed 15.04.2018]

Loureiro, M., 2015, Clean Code 101 – Meaningful names and functions, https://medium.com/coding-skills/clean-code-101-meaningful-names-and-functions-bf450456d90c. [accessed 15.10.2017]

Mytton, D., 2004, Why You Need Coding Standarts, https://www.sitepoint.com/coding-standards/. [accessed 06.10.2017]

Nograles, E., 2017, What are advantages and disadvantages of GraphQl, SOAP and REST?, https://www.quora.com/What-are-advantages-and-disadvantages-of-GraphQL-SOAP-and-REST. [accessed 05.12.2017]

Npgsql, http://www.npgsql.org [accessed 15.04.2018]

Öztekin, O., 2014, Unit Test Yaparken Sıkça Karşılaşılan Hatalar, http://orhanoztekin.blogspot.com.tr/2014/06/unit-test-yaparken-skca-karslaslan.html. [accessed 01.02.2018]

Riehle, D., 2007, Why object-oriented frameworks?, http://dirkriehle.com/computer-science/research/dissertation/chapter-1.html. [accessed 15.11.2017]

Software Framework, https://en.wikipedia.org/wiki/Software_framework. [accessed 13.11.2017]

Tasarım Örüntüleri, https://tr.wikipedia.org/wiki/Tasar%C4%B1m_%C3%B6r%C3%BCnt%C3%BCleri. [accessed 02.11.2018]

Tungut, B., 2015, SOLID Prensipleri : 01 Single Responsibility ve Agile Metodojilerde Önemi, http://www.buraktungut.com/solid-prensipleri-01-single-responsibility-ve-agile-metodolojilerde-onemi. [accessed 18.10.2017]

Yılmaz, F., 2016, Framework Nedir, Library Nedir?, https://fevziyilmaz.wordpress.com/tag/framework-cesitleri/. [accessed 11.11.2017]

CURRICULUM VITAE

Name and Surname: Yunus Emre Kırkanahtar

Permanent Address: *Zekeriyaköy Mah. Boğaziçipark Evleri Ladin1 Apt. D.18 Sarıyer İstanbul*

Date and Place of Birth: *İstanbul 14.02.1982*

Foreign Languages: *English, French*

Primary Education: *Ö.Cent Lisesi 1993*

Secondary Education: *Ö.Saint Michel Lisesi 2000*

Bachelor's Degree: *Marmara Üni. Bilgisayar ve Kontrol Öğretmenliği 2005*

Master's Degree: *Bahçeşehir Üniversitesi*

Name of Institute: *Fen Bilimleri Enstitüsü*

Name of Program: *Bilgisayar Mühendisliği*

Career:
CuteSafe Teknoloji 2016 –
MasterCard Payment Transaction Services 2011-2016 (It was called "Provus Bilisim" until 2013)
Invenoa Yazılım 2011-2013 (OutSource at Provus Bilisim)
Gurus Teknoloji 2009-2011
Maya Butik (2005-2009)
Vaillant Türkiye (2003-2005)