T.C. DOĞUŞ ÜNİVERSİTESİ

INSTITUTE OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

THE IMPACT OF STATIC ANALYSIS TOOLS ON SOFTWARE QUALITY, PRODUCTIVITY AND COST

MASTER THESIS

MEHMET YILDIZ

201195002

ADVISOR:

Assist. Prof. Dr. YASEMİN KARAGÜL

Istanbul, 2019

T.C. DOĞUŞ ÜNİVERSİTESİ

INSTITUTE OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

THE IMPACT OF STATIC ANALYSIS TOOLS ON SOFTWARE QUALITY, PRODUCTIVITY AND COST

MASTER THESIS

MEHMET YILDIZ

201195002

ADVISOR:

Assist. Prof. Dr. YASEMİN KARAGÜL

Istanbul, 2019

	YÜKSEK LİSANS TEZ SINAV TUTANAĞI	Doküman No	FR.1.26
		Yürürlük Tarihi	1.11.2017
		Revizyon Tarihi	1.11.2017
		Revizyon No	1
		Sayfa	1 / 1

SOSYAL BİLİMLER / FEN BİLİMLERİ ENSTİTÜSÜ

Tarih : 03/05/2019

Anabilim/Anasanat Dalı : Bilgisayar Mühendisliği

Öğrencinin Adı Soyadı : Mehmet Yıldız

Öğrenci No : 201195002

Tez Danışmanının Adı Soyadı : Dr. Öğr. Üyesi Yasemin Karagül

İkinci Tez Danışmanının Adı Soyadı : _____

Tezin Başlığı : The Effect of Static Analysis Tools on Software Quality, Productivity, and Cost

Doğuş Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliği'nin 32.Maddesi uyarınca yapılan değerlendirmeler sonunda;

☒ **tezin kabul edilmesine** ☐ **tezde düzeltme verilmesine** ☐ **tezin reddedilmesine**

(oy birliği) / oy çokluğu ile karar verilmiştir.Gereği için arz olunur.

Danışman Üye
Dr.Öğr.Üyesi Yasemin Karagül

Üye
Profesör Dr.
Selim Akyokuş

Üye
Doç.Dr.
Hüseyin Kılıç

Üye

Üye

Anabilim/Anasanat Dalı Başkanı Onayı:
Doç.Dr. Hüseyin Kılıç

DECLERATION

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or institution. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Mehmet Yıldız

Signuture:……..

Date : ……

PREFACE

In this thesis, the effects of static analysis tools on software quality, productivity and cost were investigated. This research includes both theorical information and evaluations of experiment results from real world's IT appliance. I would like to thank my advisor Assist. Prof. Dr. Yasemin Karagül, for her interest, support and guidance. Furthermore, I would like to express my gratitude to my dear wife Yasmin Yıldız for her valuable support during my thesis studies.

Istanbul, 2019 Mehmet Yıldız

ABSTRACT

With the spread of technology, impact and importance of software in our daily life are increasing considerably. Considering importance of software in our daily life, code quality, bug-free development and conformance for coding standards are becoming indispensable. The purpose of this work is to list the advantages for both development team and software quality by demonstrating the importance of static code analysis for the software life cycle to obtain software that is reliable, low maintenance, low-cost, standards-compliant and bug-free (with early error detection and prevention). In order to develop a standards-compliant software, it's source code must be written in accordance with the standards ruleset and analyzed carefully for conformity in the development phase of software development life cycle (SDLC). The analysis can be performed dynamically or statically. Static analysis is performed on program source code without actually executing program but dynamic analysis is performed while executing program source code. The static analysis made with automatic analysis tools produces reports about software quality. Static code analysis is often used to find potential errors, detect possible weaknesses in the program code that may lead to weak points, maintain code quality, or check compliance with coding standards. In this thesis, the Java source codes of 29 different projects developed by 5 different development teams of a telecommunication company have been evaluated by SonarQube and the outputs of this evaluation are discussed. Analysis tool automatically starts the analysis process on the project codes that is retrieved from the version tracking system (SVN) and finds the possible weak points, bugs and noncompliant issues in code sections. By correcting the findings and adding the automatic code analysis step to the development process, early error detection and preservation of the quality of the software are ensured. Detecting and correcting errors, increasing productivity, reducing maintenance cost by 21% and generating clean code before execution in the production environment are the lucrative outputs of this work. As a result of the study, the improvement reports triggered by the analysis reports not only provide the quality of the code but also increase the capabilities of the development team.

Keywords: Software analysis, static analysis, code quality, program error checking, code analysis, code review, software quality, error prevention, early error detection, analysis tools, maintenance cost.

ÖZET

Teknolojinin yaygınlaşmasıyla birlikte günlük yaşamımızda yazılımın etkisi ve önemi gün geçtikçe artmaktadır. Bu bağlamda yazılımın kod kalitesi, hatasız geliştirilmesi ve kodlama standartlarına uygunluğu vazgeçilmez unsurlar olmaya başlamıştır. Bu çalışmanın amacı, güvenilir, bakım maliyeti düşük, standartlara uygun ve hatadan(erken hata tespiti ve önleme ile) arınmış yazılım elde etmek için statik kod analizinin yazılım yaşam döngüsü için önemini ortaya koyarak hem geliştirme ekibi hem de yazılım kalitesi için avantajlarını listelemektir. Standartlara uygun bir yazılım geliştirmek için yazılım geliştirme yaşam döngüsünün geliştirme aşamasında geliştirilen kaynak kodun standart kurallara uygun olarak yazılması ve dikkatli bir şekilde analiz edilerek uygunluğu kontrol edilmelidir. Analiz, dinamik veya statik olarak yapılabilir. Statik analiz program çalıştırılmadan veya yürütülmeden kaynak kod üzerinde yapılırken, dinamik analiz gerçek olarak çalışırken yapılır. Otomatik analiz araçları ile yapılan statik analiz yazılım kalitesiyle ilgili raporlar üretir. Statik kod analizi çoğunlukla potansiyel hataları bulmak, program kodunda zayıf noktalara yol açabilecek olası zafiyetleri tespit etmek, kod kalitesini korumak veya kodlama standartlarına uygunluğu kontrol etmek için kullanılır. Bu tezde bir telekomünikasyon şirketinde 5 farklı geliştirme ekibi tarafından Java ile geliştirilen toplam 29 proje kaynak kodu üzerinde SonarQube ile çalışılarak elde edilen çıktılara yer verilmiştir. Analiz aracı ile otomatik olarak sürüm takip sisteminden çekilen proje kodları üzerinde analiz işlemi başlatılarak projelerdeki hata, zayıf nokta ve kodlama standartlarına uymayan bölümler için bulgular ortaya çıkarılmıştır. Elde edilen bulguların düzeltilmesi ve geliştirme sürecine otomatik kod analiz adımının eklenmesiyle yazılımda erken hata tespiti ve kalitenin devamlılığının korunması sağlanmıştır. Yazılımın üretim ortamında yürütülmeden önce hataların tespiti ve düzeltilmesi, verimliliği arttırma, bakım maliyetini %21 oranında düşürme ve temiz kod üretme ise bu çalışmada elde edilen önemli çıktılarıdır. Çalışma sonucunda analiz raporlarının tetiklediği iyileştirme faaliyetlerinin yalnızca kod kalitesini sağlamakla kalmayıp aynı zamanda geliştirme ekibinin kabiliyetlerini arttırması gözlemlediğimiz güzel kazanımlardandır.

Anahtar Kelimeler: Yazılım analizi, statik analiz, kod kalitesi, program hata kontrolü, kod analizi, kod incelemesi, yazılım kalitesi, hata önleme, erken hata tespiti, analiz araçları, bakım maliyeti.

TABLE OF CONTENTS

Page no

LIST OF TABLES

Sayfa no

LIST OF EQUATIONS

Page no

ABBREVIATIONS

CLEARCASE : Software configuration management tool

ECLIPSE : Integrated development environment

GIT : A distributed version control system

JENKINS : Continuous integration and continues delivery platform

LDAP : Lightweight directory access protocol

SDLC : Software development life cycle

SONARQUBE : Continuous code quality platform

SVN : Subversion is a version control system

1. INTRODUCTION

Especially in recent years, the quality of software has started to gain importance with the spread of technology. The fact that it touches every aspect of our lives and that this scope is expanding day by day brings the standards and quality to the foreground in the software. Because the domain is large and deep, the smallest error in the software causes major problems. Errors included in the software during the development process are known to cause the software not to meet expectations. Some of the negativities that may arise from software failure include endangering human life, stopping and blocking communication and transmission, erroneous production, loss of honorable customers and bankruptcy. Nowadays, it is clearly known that due to errors in software, many companies and indirectly people have been in difficult situation and suffered damage.

The building blocks of a software program consist of lines of code written by programmers. Considering the importance and impact of software in our daily lives, for each line of code, the code quality, error-free development and compliance with coding standards have become indispensable elements. Testing is known as a method to obtain information about the functionality and quality characteristics of the software and to determine the errors. However, most of the errors encountered/discovered in the testing phase are late findings. Because in the production environment and even in the testing phase, each error causes to re-run the SDLC process from previous stages and must be fixed and retested respectively. This situation means a significant waste of time and cost. Early detection of software related errors and to capture and resolve these errors in the development phase provides significant savings. According to Kumaresh and Baskaran (2010), in the development process, error prevention provides quality in software by playing an important role in improving software process quality.

As an outcome of this study, it is expected to increase productivity by improving software development process with error prevention solutions, increasing quality awareness in software and obtaining software with high quality, but lower maintenance and development costs. The objectives of this study are as follows;

- to have reliable, cost-effective, standards-compliant and error-free software

- to improve software quality by preventing and correcting errors during software development
- to improve software development process with early error detection
- to enable developers to develop in accordance with software development standards and improve their skills

In order to achieve the above objectives, 29 java projects which are developed by 5 teams in a telecommunication company have been analyzed. Aim of the analyzes were to measure the quality of the projects, to detect problems and to check the compatibility of the source code with the coding standards (Java Code Conventions, 1997; Java Style Guide, 2014). In these studies, the SonarQube (Continuous Code Quality Platform, 2017 ; Campbell & Papapetrou, 2013) static code analysis tool, which is configured to be triggered automatically via Jenkins (Armenise, 2015), is used for static code analysis.

1.1 Objective and Scope of The Work

With the rapid development of technology, the importance and impact of software in our lives is increasing day by day. The number of software products is increasing rapidly with the expansion of technology in business and everyday life. Considering this wide scope and impact, it is important that the software complies with the code quality, error-free development and coding standards. In this study, it is aimed to determine and solve the problems added by the developers during the development process, improve the quality by learning from mistakes and ensure the continuity of all these stages as a part of SDLC. At the end of this process it is aimed to increase productivity by producing high quality software, low cost in both development and maintenance. In this work, experimental studies will be done and the findings of the analysis will be examined and corrected. At the end of this work, positive and negative impacts of applying this process as a part of SDLC will be evaluated.

As mentioned in the studies by Nagappan & Ball (2005), the quality of the software is related to the error rate in the software and this data can be used as an indicator to determine the quality of the software. In this study, in the field of telecommunication, an analysis of 29 projects developed by java by 5 development teams in the software development phase was carried out with static analysis tools. The main purpose of the study is to increase the productivity by detecting the compatibility of the software with

the coding standards and the errors it contains. Only 5 projects were included in the study due to the fact that important and main projects were included in the study. For static code analysis, SonarQube static code analysis tool is configured to be triggered automatically via Jenkins (Jenkins, 2017; Armenise, 2015). The default rule list on the tool is used and the classification details in the findings are as in Table 1.1.

Table 1.1 Finding Category And Priority Order of Importance

Finding Category	**Priority**	**Description**
Blocker	1	Operational/security risk: It might make the whole application unstable in production. For example: calling garbage collector, not closing a socket, etc.
Critical	2	Operational/security risk: It might lead to an unexpected behavior in production without affecting the integrity of whole application. For Example: Null-Pointer Exception, badly caught exceptions, lack of unit tests, etc.
Major	3	It might have a substantial impact on productivity. Ex: too complex methods, package cycles, etc.
Minor	4	It might have a potential and minor impact on productivity. Ex: naming conventions, Finalizer does nothing but call superclass finalizer, etc.
Info	5	Unknown or not yet well defined security risk or impact on productivity

Source: SonarQube in Action, 2013,p.30

In the following sections, we will talk about some concepts such as bug, code smell and vulnerability. We refer them as issue. According to the sonarqube, detailed explanation of the concepts are as follow;

- Bug: An issue that represents something wrong in the code. If this has not fixed yet, it will, and probably at the worst possible moment. This needs to be fixed.

- Code Smell: A maintainability-related issue in the code. If it is not fixed, the best developers will have a harder time while they are making changes to the code. At worst, they will be so confused by the state of the code that they will introduce additional errors as they make changes.
- Vulnerability: A security-related issue which represents a backdoor for attackers. See also Security-related rules.

1.2 Method Of The Study

We have done experimental studies through existing code pools. Some reports have been generated by analyzing the findings on code automatically with static analysis tools. In order to eliminate the findings in these reports, code improvement studies were initiated in coordination with development teams.

In the development phase, analysis report creation process starts with the automatic triggering of the static analysis tool and a code analysis report is obtained at the end of the process. This report consists of the findings of the code that do not comply with the standards and it's weaknesses. The Software Code Quality Requirements document[Annex-1], consisting of 80 items, was published in order to avoid the repetition of the frequently encountered problems as a result of the investigations made according to these analysis reports. This document is based on Java coding rules published by Sun Microsystems (Java Code Conventions, 1997) and Google (Java Style Guide, 2017) and adapted to the corporate culture.

1.3 Constraints Of The Study

In this study, a limited number of java projects and a code of 436846 lines developed by a total of 5 development teams have been studied in a company operating in the telecommunications sector. Static code analysis tool was used to analyse the source code. Analyzes were made in order to measure the quality of the projects, to detect the problems and to check the compliance with the standards. In these studies, SonarQube(Continuous Code Quality Platform, 2015; Campbell ve Papapetrou, 2013) static code analysis tool which is configured to be triggered automatically by Jenkins (Jenkins, 2017; Armenise, 2015) is used. These tools are preferred because of their

practical use in the information technology sector, they are widely accepted, they are compatiable with different development environments, also java language compatible and open source. Errors in the production environment are reported by the people in charge of the system through the *application lifecycle management application* and this application has been used for reporting purposes. Findings have been obtained with experimental studies and studies have been carried out on these findings and errors detected in real production environment. The limitations of the study are as follows:

- Project codes developed using only java development language
- Data from the 5 largest projects in the 5 projects were included in the evaluation. The first 5 applications with the highest number of lines of code were selected from 29 projects. Due to time constraint, high error density in the 5 selected projects and the fact that this study is a pilot study, projects with a small number of code lines other than 5 main projects and low number of code lines are not covered in this study.
- SonarQube analysis tool used as the only analysis tool
- Work on projects developed by a limited number of teams

2. LITERATURE SURVEY

Software developed in corporate companies needs are constantly being developed to meet additional development demands coming from frequently developed software. The existence of a software's continuous development cycle, new development needs and maintenance costs increase the future costs of that software. The fact that the software does not comply with the coding standards and therefore has low quality causes maintenance costs to increase more deeply. In order to ensure the quality in the software, to be controlled and to ensure the continuity of the process, academic resources have been examined and short summaries are given below for the important sources. In recent researches (Nagappan ve Ball, 2005; McConnel, 2001, s.5-7; Jalote ve Agarwal, 2007; Fagan, 1976, s.186; Emden ve Moonen, 2002; Catal, 2011) it has been observed that publications are mainly related with detection of errors but studies on error prevention and software quality process improvements are limited.

McConnell (2004) is a comprehensive publication on the basics of quality in software, what characteristic features are the quality indicator in software, and techniques are proposed to improve the quality of the software. According to Mcconnel, “trying to improve the quality of software by increasing the frequency and amount of the test is the same as falling into the misconception that you will lose weight more frequently when weighed. If you want to lose weight, do not buy a new scale; change the diet. If you want to improve your software, do not test further; Focus on better development”. The results of the studies are important points obtained in terms of error detection and inspection. According to the researches, complexity analyzes with static analysis tools increase the productivity in the maintenance process by approximately 20% (McConnel, 2004). Another study by McConnel(2001) the cost of a defect or error detection has grown exponentially over time and that the cost of fixing errors in early life is cheaper. Code review methods and automatic tools are recommended for early error detection (McConnel, 2001).

The study by Jalote and Agarval (2007) investigated the effect of error analysis as a feedback mechanism to improve quality and productivity in an iterative software project. The analysis of the errors in a iteration includes the examples of its use and benefits in a commercial project, which may lead to improvement in quality and

productivity by providing feedback in the subsequent iterations to prevent errors. It has been determined that the failure rate of the following iterations is lower than the previous iterations. As a result, it is stated that such an analysis will be performed at regular intervals in large-scale projects and this will increase the quality of the software and increase productivity. Similarly, the experience gained in other projects is an emphasis on the fact that structured feedback from a repetition is very effective in improving quality and productivity in future iterations or new projects (Jalote and Agarwal, 2007).

This study, written by Kumaresh, and Baskaran (2010) on software process quality improvement and error analysis. An error prevention tool was developed and the results obtained for this tool were shared. According to the results of the study, it was determined that the error prevention application was developed by the software developers to learn the mistakes and to learn the mistakes of others. In addition to this, error prevention work is the main result of improving product quality while decreasing product cost by decreasing development time and cost, decreasing customer satisfaction and need of rework. It is emphasized that the quality of the software provides an important role in the development of software process quality by preventing error in development process (Kumaresh & Baskaran, 2010).

2.1 Software Quality

The concept, known as software control or software review, was first introduced in 1976 by Fagan (1976, pp.185-202) and was used to improve software quality. The software review involves checking the software code, design and documentation for the detection of potential potential problems (Emden & Moonen, 2002). Detection of potential problems that are considered to be quality indicators of the software is only possible with code analysis. In systems developed with object-based architecture, error discovery rate and distribution is the key indicator for the quality of the software (Booch, 1998; Nagappan & Ball, 2005). In order to predict the errors in the software, it was emphasized that this process should be automated (Catal, 2011).

The quality of software consists of two basic concepts: internal characteristic and external characteristic. External quality characteristics of the software are generally the result of the reflection of the internal characteristics. A quality software can be used with

its external features, accurate, efficient, reliable, in integrity, adaptable, consistent and robust, while it must be sustainable, flexible, portable, reusable, readable, testable and understandable with its internal features (McConnell, 2004; Bourque and Fairley, 2014).

In order for a software to meet the basic quality requirements, it must be developed in accordance with the coding standards, free of errors and thoroughly tested. Before testing, the software must go through a detailled examination to determine that it is compliant with coding standards and does not contain errors. It is not possible to diagnose and fix the errors of the software without examining the program's code about whether the software contains coding errors or not. Although it is possible to reveal these errors by conducting code reviews through review meetings, it does not bring an effective result in terms of applicability, continuity, time and cost. In addition, experience, domain knowledge and focus of the person or people participating in the code review are serious problems that can vary in the implementation and continuity of this method.

The software is like a prototype moving on the production line into a product. A fault that is recognized in the next step requires that the product to return one or more steps and take corrective actions or actions. In order to ensure the quality of the software and to maintain the continuity of error-free development early error detection is very important. Because, in the production environment and even in the testing phase, when an error is detected and the correction of this error is a costly situation because it requires all stages to be run again. Diagnosing and resolving development-related errors during the development phase provides significant time and cost savings. As in other areas, the preventive approach serves to eliminate errors by early intervention, to reduce costs and to pass through the quality control indicators of the work outputs. In this way, the quality and quality continuity of the software can be maintained.

2.2 Coding Standards

The most important quality indicator of the software is whether software complies with the coding standards or not. Coding standards are a set of rules that recommend the most appropriate coding practices for each aspect of a program written in that language for a particular programming language. Any software has to be developed in accordance with the coding standards to meet expected quality. Especially in large-scale software

projects, compliance with these standards represents the quality of those projects. During a code maintainence task, developers often spend most of their time trying to understand the code. The most important reason for this situation is the code pools developed without caring the standards. It is very difficult to modify codes with low readability, complex conditional logic, and repetitive logic (Fowler, Beck, Brant, Opdyke & Roberts, 1999). Applying the coding standards during the development phase enables to produce understandable code and then the time spent to understand the code is significantly reduced. One of the key elements of a development plan that needs to be followed to obtain a quality software is the coding standards (Kevitt, 2010).

The set of rules in the coding standards are structural quality guidelines for software. Software developers must follow these guidelines to improve the readability of the source code, to facilitate software maintenance, and to develop software that conforms to the standards. Since these standards are not mandatory by the compilers, static analysis tools are used to understand whether the standards are complied with.

Coding standards make it easy for everyone in the company to understand each other's codes and work together comfortably. If the code is not written and edited according to the programming instructions, it is very difficult to develop, integrate and maintain on a small part of the software, especially in crowded development teams. It is not always easy to implement compliance with coding standards. While all developers involved in the project have an obligation to know and evaluate the software guidelines, such as time constraints, people's reaction to limiting or not accepting the rules, these are the main factors that make compliance with these guidelines difficult.

In conclusion, ensuring that the code complies with the coding standards improves the code quality. When this process is supported by automatic conformity check and the project specific code smells are allowed to be detected, automatic smell sensing becomes a conformity control process (Emden and Moonen, 2002).

2.3 Code Smells

The code smell, also known as malicious odor in the software code, points to a problem in the source code of a program, or a more profound problem that is likely to be severe. According to Fowler, Beck, Brant, Opdyke and Roberts (1999); code smell is

often the surface indicator that corresponds to a deeper problem in the system. One of the main reasons behind the smell of code is the errors in development and design process. In particular, the failure to comply with coding standards during the development phase significantly reduces the quality of the code and results in shorter software life. Code smells which are pointing defects and errors in the code represents the quality of software systems adversely. They also effect both flexibility and maintenance effort causes serious problems and difficulties. The common smells in the code can be listed as follows (Fowler, Beck, Brant, Opdyke & Roberts, 1999):

- Duplicated Code blocks
- Large class, method and function: High number of code lines in class, method and function
- Complex methods or classes: Code blocks that contain nested conditions and loops. Overall, the increase in the number of methods in a class makes it more complex and application-specific (Booch, 1998).
- Long parameter list: Number of method and function parameters (Maximum 7)
- Dependent change chain: A change affects multiple points and needs a dependency change
- Unregistered code blocks in the class (Feature envy). The code block is related to a different class than the class it is in.
- Repetitive data sets: Most of the time there are three or more data items at more than one point. For example, the fields of several classes, parameters in many function signatures.
- Use of unnecessary complex data structures (primitive obsession): preferring special classes / complex data structures or objects instead of primitive data types.
- Use of unnecessary switch, condition structures: Use of duplicate switches, condition and control structures
- Parallel inheritance hierarchies: The requirement to create a subclass of another class every time you create a subclass of a class
- Unused class, method, field, parameters: Unused code fragments, class, method, field or parameter stacks
- Missing library class: Frequently used and lack of class in the library

- Comments: Comment lines written to describe complex and unclear code structures. When you feel the need to write a comment, try reconfiguring the code first, so any comment is unnecessary (Fowler, Beck, Brant, Opdyke & Roberts, 1999).

The design flaws are known to have negative effects on the flexibility and maintenance of the software systems (D'Ambros, Bacchelli & Lanza, 2010). ; However even if the design is correct there should not any code smells in the code. If the coding standards are not followed at the point where the design is put into practice, it is inevitable that smells will be formed in the code.

3. RESEARH METHOD

With the recent development of DevOps (DevOps, 2017) and Agile (Agile, 2017) software development methods, the software continues its life cycle with successive versions as a result of rapid development and release cycles. These fast and small changes can cause problems in the big picture while the software is following a difficult change process (Goldstein & Mount, 2015). In the development process, if a fault is not detected and not corrected, it will remain a danger for the life of the software. The longer an error lives in the software, the greater cost to solve the error will be required. According to McConnell (1996), if you do not do the job at the time of development that may cause an error in software, then you should take 10 times more time to correct this error (McConnell, 1996, p.45).

When an error is detected at the beginning of the development cycle, it is generally known that the cost of repairing the error is very low. Detecting the fault at this stage is of great importance. According to Emden and Moonen (2002), one of the advantages of software review is to analyze the software before testing. This allows the problems to be identified while at the development stage and to be solved at a very low cost (Emden & Moonen, 2002). One of the main motivation to start this study is to have quality software by reducing the cost of end-to-end software and maintenance effort. In addition to this we want to raise awareness of not risking the future of the software by ensuring the quality of the software, compliance with the standards, early detection of errors and taking necessary actions. Because the life, cost, quality and future of the software is just as important as not being left to the initiative of employees in development and test teams.

In this study, the following questions will be searched in order to ensure the quality of the software and maintain its continuity:

I. How can quality be measured during software development?
II. Can a software error prevention system be installed? What benefits do you have if it is installed?
III. Is it possible to detect a possible problem early in the software development stage by catching some problems before it reaches the test stage with an early diagnosis

without being faulted and to correct the problems that are caught in the test phase? How should it be done if possible?

IV. Code quality control, compliance with standards, code error tracking or code review can be done automatically?

V. How effective are automated code analysis tools in reducing maintenance costs?

3.1 Analysis Tools and Their Contributions To Software Quality

The technical debt refers to the correction of the design or coding of the software, or the tasks that need to be done to make the coding complete and accurate (Campbell & Papapetrou, 2013, p.23). Static analysis is a technique used to identify and analyze software properties from source code; these can be defined as items such as packages, classes, relationships, code lines, errors, complexity, coding violations, and others.

Until recently, the analysis tools were weak in terms of their primitive and in-depth analysis capabilities and could not take place in the software development process. However, recent assessments of newly developed and updated existing vehicles have shown that these tools have contributed more than expected to software development process. According to the researches, complexity analyzes with static analysis tools increase the productivity of care by 20% (McConnell, 2004, p.778).For example another study, it is emphasized that productivity will go to zero as the complexity of code increases over time (Martin, 2008, p.4). These tools can analyze the structure of a program and suggest reconfiguration that could improve this structure (Fowler, Beck, Brant, Opdyke & Roberts, 1999). In fact analysis can be done dynamically or statically. Static analysis is performed on the source code without running or executing the program; In general, static analysis responds to a wider range of questions than a dynamic analysis (Bush, Pincus & Sielaff, 2000). In the development period of a software, early errors and correcting errors not only reduce the cost but also provide error-free software. A software should be analyzed for validation even if it is developed by expert programmers. Because the results of the analysis show that the professional programmers may have some problems in their codes.

Applying unit tests, system tests, quality assurance or manual code inspections, most errors in code level still cannot be detected. In order to detect these errors in the development phase auto-runnable static analysis tools developed. By the help of these

tools, detecting defects become quickly, analysis time is shortened, error is detected during the development phase, and on this occasion significant savings are provided in software development life cycle.

Static analysis tools are able to identify the following conditions (Continuous Code Quality Platform, 2017; Bougroun, Zeearaoui & Bouchentouf, 2016; Campbell, & Papapetrou, 2013, p.13-18):

- Program code does not comply with the rules
- Parts of the program that may interfere with the correct operation,
- Some points that do not obey the rules that hinder some non-functional quality aspects such as maintenance feasibility and complexity
- Non-compliance with the best and safest programming methods
- A large number of topics aimed at traditional (manual) controls
- Piles of dangerous code
- Security-critical code sections

The root cause of critical and blocking errors is the result of bad habits in programming and directly affects the technical debt. The basic approach to using SonarQube as a static analysis tool is to propose to developers the use of standards and maintain a reference criterion that does not increase technical debt, especially when working with development teams in crowded organizations. The main reasons for choosing SonarQube analysis tool in this study are as follows (Campbell and Papapetrou, 2013):

- The tool is famous for its controlling the quality and standards of Java projects.
- Plugin support for Findbugs, pmd, checkstyle, cobertura, etc
- Supporting 20+ software development languages. It has large scope from code smells to security vulnerabilities.
- Supporting open source platform, quality-analysis experts recommend it
- It is compatible with continuous integration tools (eg jenkins, bamboo, Hudson)
- It can work with Sonarlint plug-in with integrated development environment tools (eclipse, visual studio, intellij-idea etc.) and to guide the developer to the standards
- It is accurate and consistent in the work done and one step ahead of its competitors in issues such as community support behind the tool
- It is able to run as fully automated analysis and integrations

- It supports integrations with external systems such as LDAP, Active Directory, SVN and GitHub
- Automated and continuous monitoring of code quality with this tool
- To be able to produce reports about duplicated code, unit test status, comments, errors, vulnerability, code scope and code complexity
- Supports SQALE rating. The quality classification of projects with SQALE (Letouzey, 2016) rating is directly related to the technical debt ratio of the project and the Technical Debt Rate is can be described as follows (Letouzey, 2016):
  - Technical debt of your project (= debt of all problems)
  - Divide by re-write cost estimate(re-write project from scratch)

3.2 Software Quality and Static Analysis Tool

While the quality of the software is based on the internal and external characteristics of the software, the external quality characteristics of the software are generally the result of the reflection of the internal characteristics (Lincke, 2007). In this study, the source code reflecting the internal characteristics of the software was analyzed and improvement suggestions were obtained. The analysis with the static analysis tool the code review process can produce analysis reports by working manually or schedule to run automatically. The result of the analysis includes the quality level of the software. This level is named as Quality Gates. According to the criteria determined as Quality Gate, tool reports the result as passed or failed. Since these criteria consist of parameters such as newly added blocker / critical number of findings, code coverage rate, degree of maintenance availability, and safety assessment, the quality of the software can easily be measured in development stage. Thus, This report provides information about the quality level and the compliancy of coding standards. As a result, the code review process is performed automatically without any effort and provide information about the quality of the software and compliancy with the coding standards.

With the analysis tool used in this study some problems can be detected without entering into the test phase. Fixing the problems within development period enables blocking an issue to turn into error. In this way, most of the findings identified in the testing process in the old process do not need to be re-developed after development-> testing. This has reduced the repetitive workload on both the test and development sides.

Of course, not all of the errors encountered in the software, it is observed that the errors that can be detected with the static analysis tool are possible to be diagnosed early and the effort spent on errors in this context can be saved.

The development leader, software architect, project manager and product responsible examined static analysis reports. In addition, it is concluded as very beneficial to use during the development phase as early error detection mechanism and code inspection tool. Thus, the detection and correction of code fragments that do not meet the standards that may cause errors can be made at the beginning of the process. This provides a reduction in the number of repetitive efforts on the development and test source side.

3.3 Method And Detail Of The Work

In the study, we performed automatic analysis studies on the project source codes with the static code analysis tool to check whether the software development teams complied with the coding standards and to evaluate the software quality. The number of findings based on project-based analysis and number of findings that are resolved after compliance with code standards are as in Table 3.1

According to the results of the project-based analysis, it is possible to identify the most frequent and dense errors as a result of the number of findings, type and the number of solved findings. In the light of these data, we provided feedback to the relevant development team by taking into account the error type, importance level, distribution rate, and by providing information with code samples. Therefore, development was achieved and awareness was created in order not to repeat the same kind of findings. In this study, it was supported to identify the findings and correct them by the developers and to produce better quality outputs by taking the lessons from their own mistakes. However, each line of code analyzes by the analysis tool helped keeping the projects' source code compatible with coding standards. In addition, it provides significant gains for the future of the organization because it enables the newcomer to adapt faster and improve self-correction skills that yield to increase the quality to the next level.

Table 3.1 Analysis Issue Details

Project (# of code line)	Issue Type	Total Issue Count	Issue Detail/Explanation	Fixed Issue Count
ProjectA (61410)	Blocker	12	8 Null pointers should not be dereferenced 2 Throwable and Error should not be caught 2 Conditions should not unconditionally evaluate to "TRUE" or to "FALSE"	2
	Critical	97	94 Exception handlers should preserve original exception 1 Exit methods shouldn't be called 2 Lack of multi-threading concept usage	2
	Major	1811	**	25
ProjectB (117979)	Blocker	913	11 Null pointers should not be dereferenced 274 Throwable and Error should not be caught 628 Conditions should not unconditionally evaluate to "TRUE" or to "FALSE"	890
	Critical	836	483 Fields in a "serializable" class should either be transient or serializable 342 Exception handlers should preserve the original exception 11 IndexOf checks shouldn't be for positive numbers	828
	Major	5825	**	4394
ProjectC (40901)	Blocker	323	281 Conditions should not unconditionally evaluate to "TRUE" or to "FALSE" 4 Access information or credentials shouldn't be hard coded 1 Throwable.printStackTrace(...) shouldn't be called 1 Static fields should be final 36 Throwable and Error should not be caught	*
	Critical	527	509 Fields in a "serializable" class should either be transient or serializable 8 Exception handlers should preserve original exception 10 Equality tests should not be made for floating point values	*
	Major	2644	**	*
ProjectD (136185)	Blocker	525	487 Conditions should not unconditionally evaluate to "TRUE" or to "FALSE" 22 Throwable and Error should not be caught 16 Null pointers should not be dereferenced	498
	Critical	500	318 Fields in a "serializable" class should either be transient or serializable 18 Equality tests should not be made with floating point values 164 Exception handlers should preserve original exception	489
	Major	5843	**	2681

Table 3.1 (Continued)

Project (# of code line)	Issue Type	Total Issue Count	Issue Detail/Explanation	Fixed Issue Count
ProjectE (80371)	Blocker	467	328 Conditions should not unconditionally evaluate to "TRUE" or to "FALSE" 78 Resources should be closed 30 Null pointers should not be dereferenced 23 Jump statements should not occur in finally blocks 4 Throwable and Error should not be caught 2 The class overrides “equals()” and should therefore also override “hashCode()” 2 equals() should test object type	323
	Critical	664	327 Fields in a “serializable” class should either be transient or serializable 176 Throwable.printStackTrace(...) shouldn’t be called 17 if /elseif statements shouldn’t contain the same conditions 42 Exception handlers should preserve original exception 86 Common static fields should be final 8 Access information or credentials shouldn’t be hard coded 8 The same operands shouldn’t be used with the same operator	270
	Major	3826	**	1531

* The project is not included in the improvement process. Since the project will be retired.

** Major faults often encountered::

- Complex methods, function or class
- Nested checks, loops, vb.(if, for, while, try, switch)
- Large class
- Unused method, variable, field, class, local variables or method paramters
- Casting primitive data types
- Unauthorized access to unprotected class
- Nested try-catch blocks
- Unnecessary assignments and duplicated code
- Unnecessarily usage of asynchronous object usage (Using StringBuffer instead of StringBuilder)

In the 5 projects discussed within the scope of the study, a study was carried out for the errors detected in the production environment between 01.01.2016-01.01.2017 and followed on the application life cycle management(ALM) application. The aim of the error analysis in this study is to find the ratio of errors that can be determined by static analysis tools to all errors. Thus, if these tools are used in the development phase, the effects on the cost are calculated and their positive and negative aspects are evaluated. The production errors determined based on the project and the analysis findings detected in the source codes of the versions where these errors are received are as follows. Detailed information on the errors detected with the analysis tool is given on Table 3.2.2. Since

the aim is to determine how much effort is spent on the errors that can be found with the analysis tools, the error details are not included since other errors could not be found with the analysis tool.

Tablo 3.2: Effort Details for Detected Errors in Production Environment

Project Name	Error Severity	# of issues	Detected by analysis tool	Paid Effort (man/day)	Error Detail	Total Effort ** (man/ day)	Effort Saving Rate(%) ***
ProjectA	High	1				86	9/86*100 = 10.46
	Medium	3					
	Low	6	1	9	Error while updating XX item. An unhandled exception occurred.		
ProjectB	High	1				102	24/102*100 = 23.52
	Medium	1					
	Low	2	1	24	Maximum number of connections exceeded. Restart required because of performance problem.		
ProjectC	High	*	*	*	*	*	*
	Medium	*	*	*	*		
	Low	*	*	*	*		
ProjectD	High	2	1	2	Exact fetch returns more than requested number of rows.	44	(2+2+5+1) / 44 *100 = 22.72
	Medium	7	2	2+5	2: Causes to stop the process at the ProjectD. 5:When pressed the button to “show the item”, then “An Error Occurred" exception was detected. Ex handling		
	Low	2	1	1	“Function must return a value” exception was detected.		
ProjectE	High					87	(2+2+20)/87 *100 = 27,58
	Medium	8	3	2+2+20	X user cannot logout error. User hangs logged in and bash process cannot be completed.		
	Low	2			Y connections cannot logout. In addition, after logout due to timeout, telnet connection cannot be re-establisted. Causes to exceed max. number of connection.		
						319	21

* Due to the decision of retirement of the ProjectC, it was not included in the analysis and improvement studies and the findings were not corrected.

** Total Exercise (man / day): is the total of man / day spent in the production environment (without the analysis tool) to correct all detected errors.

*** Effort Saving Rate (%) = (Total Effort Spent on Errors Detected by Analysis Tool) * 100 / (Total Effort Spent for All Errors)

4. EXPERIMENT RESULTS AND FINDINGS

From the findings of the analysis, work plan was prioritized in order of priority, and tasks were initiated to correct blocking, critical and high priority findings by development teams, respectively. As a result of the corrections, the reduction in the number of findings and the improvement in the code quality yielded the following benefits:

- Ensuring that the software to be developed is made in accordance with the coding standards
- Elimination of hidden findings and errors (48%) in the project codes developed within the last 5 years
- The development team to learn from the wrong and to make the development of the same error without re-making (Because a mistake and correction action teach someone not to do the same error again.). For example; As a result of the efforts to correct the findings because of the awareness of the original exception not to be crushed by the whole development team, it was reported by the operation team that the error root cause analysis resulted in a short time. They remarked that they could detect the root cause analysis within 7-10 minutes with original exception logs.
- Developing software that does not contain any critical issue.
- Decrease in maintenance costs by 21% due to clear, readable and clean code improvements.
- Decrease in future development costs by having clean code and eliminating code readability and key-programmer dependence problems.
- Decrease in error correction requests from the test to the development team with the decrease in the number of findings in the test. For example; When some errors with the help of static analysis are considered in the development phase, the test team does not spend time to find these errors.

According to the data obtained in this study, by using static analysis tools in software development process, errors can be detected early and this reduces maintenance costs by about 21%. An additional effort requirement was not taken into account since the findings identified during the development phase of the analysis tool should be carried out within the development period of the necessary corrections before the test phase was started.

4.1 Evaluation and Feedback Meeting Notes

Evaluation interviews were organized in order to evaluate objectively positive and negative aspects of using static analysis tools. In the interviews conducted with 5 development team leaders, 3 project managers and an architectural team manager, the following questions were asked about the positive / negative aspects of these studies and the answers were summarized under positive and negative aspects:

Feedback Meeting Questions:

1. What is your opinion about the static analysis work carried out during the software development phase? In addition, how should it be operate it as a quality measurement and control point?
2. Do you find it useful to work with the static analysis tool(sonarqube) for detecting and automating a finding that was included in the development phase before it becomes a fault?
3. During the development phase of the software, is it possible to detect some possible problems before they come to the test stage? Before the test phase is it possible to detect and fix them in the development phase, how should it be done if possible?
4. Is it possible to do following actions automatically;
   - checking code quality
   - checking coding standards compliancy
   - detection of code error
   - performing code review
5. How effective is automatic code analysis tools at the point of reducing maintenance costs?
6. What do you think about the positive and negative effects of static analysis?

4.2 Positive Impacts

4.2.1 Improvement of development teams, high quality output and productivity

1. If an error was detected and fixed in the development phase, the same type errors are not reproduced with the help of the static analysis tool. Since the tool notifies

developer about problematic code when writing through integrated support with Eclipse.

2. Informing and educating the developer with the data presented by the analysis tool together with the examples that are appropriate and non-compliant with the standards.
3. Providing quality control point by showing how development teams can write code compliant with standards.
4. Because of the problems that may arise in the codes of the development experts, it is necessary to take lessons from the mistakes within the team and to spread the tradition of developing code according to the standards.
5. Because all codes are analyzed, developers are beginning to write code more carefully.
6. The fact that new experts participating in the development teams work in the code pool according to the standards increases the person's dedication, code ownership and job satisfaction.

4.2.2 Quality measurement, automatic review and early error detection

1. To be aware of an error in the development phase that may cause a problem, early intervention and, as a result, to carry out development activities that are more appropriate to the standards.
2. Using analysis tools help improving developers skills, experience and self-confidence (indirectly).
3. The code is checked before development tests, including unit testing in the development phase is reducing the number of errors captured in tests. The cost of retesting is significantly reducing after the correction within development phase.
4. It shortens the development time required for additional development and maintenance of code pools, which pass through the analyzes successfully, as it controls compliance with the standards.

4.3 Negative Impacts

1. Examination of the analysis reports; some findings need to be checked at architect / team leader / expert level

2. Loss of time for analysis and updating the rules for false-positive findings in order not the tool to repeat these findings.
3. Effort for installation and configuration of new projects on the tool and in the person's own development environment
4. Additional time request for correction of findings as a result of analysis because it is considered as extra effort and time loss by the developers.
5. Loss of sustainability due to lack of control and sanction pressure unless it is put into software development process.
6. Some of the findings by the development experts are not accepted as errors. For example, the fact that some analysis findings related to exception handling are not considered as errors.
7. There is a risk of malfunctioning of the code that runs properly during the problem fix operation.

5. DISCUSSIONS, LIMITATIONS AND THREATS

In this study, it is aimed to establish quality control structure and decrease maintenance costs in order to ensure development according to coding standards. The findings were obtained in the projects which were not compatible with coding standards and not used any analysis tools. Therefore, the decrease in maintenance costs by 21% may vary depending on the project, the analysis tool used, the capability of the development team, the coding habits and compliance level for coding standards. In addition, because of the failure to comply with the coding standards of the errors determined by the analysis tool and the assumption that it will be corrected within the development period, the cost of exertion will not be taken into account here.

The acceptance of the analysis findings by the development experts, the desire and motivation to develop in accordance with the standards are the basic requirements for the success of these studies. For example, it is necessary to ensure that the whole team does not repeat the same mistake by supporting the team in taking the lessons from mistakes and that there must be sanction to ensure this. All stakeholders need to agree and support the implementation of new code acceptance to the production environment with a prerequisite for submission of analysis reports without any evidence. In order to ensure the effective participation of all stakeholders and to manage the process successfully from end to end, the management support is critical for the applicability of the process.

6. CONCLUSIONS

Errors encountered in the production environment or in the testing phase can cause many negative consequences because the software is in use or ready to use. In general, the cost of taking pro-active (foresight) measures and fixing issues at the development phase before the problems have not yet turned into a fault remain very low comparing with the cost of correction after the problems encountered as errors.

With this approach, a study has been initiated to ensure quality in software, early detection, correction and prevention of errors in order to obtain reliable, reduced maintenance costs, standards-compliant and fault-free software. In this thesis, in the field of telecommunication, an analysis of 29 projects developed by java by 5 development teams (in the software development phase) was carried out by using static analysis tools. The main purpose of the study is to increase the productivity by detecting the compatibility of the software with the coding standards and the errors it contains.

6.1 Quality Measurement And Automation During Software Development

In the studies we have done, it has been observed that with the use of static analysis tools in the software development phase, information about the compliance of the code with the standards, and the quality level can be easily obtained. In particular, with the integration of software development editors and instant analysis tools it is possible to see instant scan of the code and get findings at the time of coding. After development phase, in order to analyze the codes in code warehouse (svn, git, clearcase, etc) we easily automated the analysis process by using continues integration tools such as Jenkins. In this way, it is possible to automatically analyze the developed code and generate analysis report of source code as quality output.

6.2 Software Error Prevention System: Early Issue Detection And Benefits

In the scope of this thesis, it is experienced to detect the errors early by the static analysis studies before testing phase or deployment to the production environment. In our experiment, firstly, the percentage of the findings opened as defect in the production

environment was investigated by using the static analysis tools. Secondly, evaluations were made on the intersection of the production defects and captured issues by analysis tools. were used by calculating the effort to eliminate errors that occurred in living environment. In the evaluation, the rate of effort saving with the use of analysis tools was determined by the following formula (Eq. 6.1).

$$\text{Effort Saving Rate } (\%) = \frac{(\text{Total Effort Spent on Errors Detected by Analysis Tool}) * 100}{(\text{Total Effort Spent for All Errors})} \qquad \text{Eq. (6.1)}$$

In this study, we have studied how many of the records opened as defect in the production environment can be detected as issue by static analysis tools. Then, the evaluations were made on the use of static analysis tools by regarding the paid effort to eliminate the errors that occurred in the production environment. Here we used real production errors and their fixing cost in terms of man/day.

6.3 Auto Analysis Tools And The Impact On Maintenance Cost:

According to the findings obtained in this study, after static analysis tools included in software development process, maintenance costs are reduced by 21%. The experiment result in terms of the contribution of static analysis tools to the efficiency, increase in productivity and quality in software are also supported by the results of studies conducted by Jalote and Agraval (2007), Kumaresh and Baskaran (2010), Mcconnel (2001) and (Mcconnel, 2004).

In the software development process, it is a great gain to identify the problems included in the software by the developers at the development stage. However, during the development phase, it was possible to fix these findings, and obtain the quality by learning lessons from the mistakes and ensure the continuity of all these processes. As a result of these processes, it has been observed that firms can significantly reduce maintenance and development costs. Some of the important contributions of the static analysis in the software development process are improving the developer's skill, increasing the readability (by having standards compliant and clean source code), decreasing the maintenance cost, directing to development according to the standards and decreasing the maintenance cost with early detection of issues. With all of these contributions, it is

recommended that static analysis should be applied at the development stage of the software to reduce the cost of effort in development, test, correction and distribution processes. However, as with most disciplines, tools and techniques only benefit if you use them at regular intervals.

In addition to static analysis, it is necessary to evaluate the results obtained from both static and dynamic analysis. With the combination of on-time analysis (which is also known as dynamic analysis) and static analysis may guide the development process to be more accurate and stable. Using the only static analysis may not be sufficient to prioritize and address issues that are high importance. Because the static analysis is carried out without actually running the program, the dynamic analysis analyzes the program during the study and can detect the findings especially at the points in the program flow. In static analysis, while the program can be monitored in a dynamic analysis, the program is monitored in a single way during the execution, which can be more efficient in comparison with static analysis (Ishrat, Saxena & Alamgir 2012).

REFERENCES

Agile(2017), http://agilemanifesto.org. Access Date: 30 Mayıs 2017.

Armenise, V. (2015). Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery. Proc. *IEEE/ACM 3rd Int. Workshop Release Eng*. Erişi adresi: https://dl.acm.org/citation.cfm?id=2820701.

Booch G. (1998). *Object-oriented analysis and design with application. 2nd edition.* , Santa Clara, California: Addison-Wesley.

Bougroun, Z., Zeearaoui, A., Bouchentouf, T. (2016). Comparative Study of the Quality Assessment Tools Based on a Model: Sonar, Squale, EvalMetrics. *Journal of Computer Sciences*. Access address: https://pdfs.semanticscholar.org/bd41/006b17e948a38f1e0bfeffe3f72de678c753.pdf.

Bourque P, Fairley E. R. (2014). *Guide to the software engineering body of knowledge (SWEBOK). version 3.0.* Los Alamitos, CA: , IEEE Computer Society Press.

Bush, W. R., Pincus, J.D., Sielaff, D.J. (2000). "A Static Analyzer for Finding Dynamic Programing Errors, *Software-Practice and Experience*, 20, 775-802. Access address: https://ieeexplore.ieee.org/abstract/document/7202950.

Campbell, G., Papapetrou P. (2013). *SonarQube in Action*. Greenwich: Manning Publications Co.

Catal, C. (2011). Software fault prediction: A literature review and current trends, *Expert Systems with Applications*, 38, 4626–4636.

D'Ambros, M., Bacchelli, A., Lanza M. (2010). On the Impact of Design Flaws on Software Defects. In QSIC'10: *Proceedings of the 2010 10th international conference on quality software*. Washington, DC. Access address: https://ieeexplore.ieee.org/abstract/document/5562941. .

Emden, E. V., Moonen, L.. (2002). Java quality assurance by detecting code smells Proc. *9th Working Conf. Reverse Engineering, IEEE Computer Society*, Access address: https://ieeexplore.ieee.org/abstract/document/1173068.

Fagan, M. E.. (1976).Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15, 3, 182–211. Access address: https://search.proquest.com/docview/222415339?pq-origsite=gscholar.

Fowler, M., Beck, K., Brant, J., Opdyke, W. Roberts, D. (1999). *Refactoring: Improving the Design of Existing Cod*e. USA: Addison Wesley Professional.

Goldstein, M., Mount, C. (2015). Automatic and continuous software architecture validation. *Proceedings of the 37th International Conference on Software Engineering*.Florence, Italy. Access address: https://ieeexplore.ieee.org/abstract/document/7202950.

DevOps(2017), https://devops.com. Access date: 30 May 2017.

Eclipse(2017), http://www.eclipse.org/. Access date: 30 May 2017.

Ishrat M., Saxena M., & Alamgir M., Comparison of Static and Dynamic Analysis for Runtime Monitoring, 2012.

Google Java Style Guide(2017), https://google.github.io/styleguide/javaguide.html. Access date: 10 April 2017.

Jenkins(2017), https://jenkins.io. Access date: 30 May 2017.

Continuous Code Quality Platform(2017), https://www.sonarqube.org. Access date: 30 May 2017.

Jalote P., Agarwal N. (2007). Using Defect Analysis Feedback for Improving Quality and Productivity in Iterative Software Development., *In proc- ITI 3rd International Conference on ICT*. Access address: http://people.cse.iitd.ernet.in/~%20jalote/papers/DefectPrevention.pdf.

Java Code Conventions (1997). Sun Microsystems, 2550 Garcia Avenue, California U.S.A. Access address: https://www.oracle.com/technetwork/java/codeconventions-150003.pdf.

Kevitt, M. (2010). *Best Software Test & Quality Assurance practices in the project life-cycle: An approach to the creation of a process for improved test & quality assurance practices in the Project life-cycle of an SME*(Master Thesis). Access Address: http://rian.ie/en/item/view/40440.html.

Kumaresh, S., Baskaran, R. (2010). Defect analysis and prevention for software process quality improvement. *International Journal of Computer Applications(IJCA)*, 42-47 Access address: https://www.ijcaonline.org/archives/volume8/number7/1218-1759.

Letouzey J. Louis (2016). The SQALE Method for Managing Technical Debt. *2012 Third International Workshop on Managing Technical Debt (MTD)*, Zurich, Switzerland. Access address: https://ieeexplore.ieee.org/abstract/document/6225997 .

Lincke, R. (2007). *Validation of a Standard and Metric-Based Software Quality Model: Creating the Prerequisites for Experimentation*. (Master Thesis). Access address: http://arisa.se/files/L-07.pdf.

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall PTR.

McConnel S. (2001). An ounce of prevention, IEEE Software, s. 5-7.

McConnell, S. (1996). *Rapid development: taming wild software schedules*. Washington: Microsoft Press, Redmond.

McConnell, S.(2004). *Code Complete. 2nd edn*. Redmond: Microsoft Press, s. 514-531, 778.

Nagappan, N., Ball. T. (2005). Static analysis tools as early indicators of pre-release defect density. *In Proceedings of the 27th international conference on Software engineering*. St. Louis, MO, USA. Access address: https://dl.acm.org/citation.cfm?id=1062558.

ANNEX

Annex I: Software Code Quality Requirements

SOFTWARE CODE QUALITY REQUIREMENTS

The software is defined as the ability of the system or its components to perform the functions expected in a given environment within a certain time frame. This document is intended to provide quality in the software developed in-house or in the custom-made manner. The requirements for quality assurance are listed below.

The requirements for code quality are grouped under 4 headings.

1. Reliability
2. Performance Competence
3. Maintenance Feasibility
4. Security

1. Reliability Requirements:

1.1. Try, Catch, Finally, exception, process or such error handling blocks should not be empty.

1.2. Exceptions catched with catch blocks should not be thrown out without any action. Exceptions must be be logged.

1.3. Functions and procedures that contain insert, update, delete, create table, or select commands to run in the database must perform exception handling.

1.4. The classes that implement the serializable interface must also perform the serializable method for the class's own and all serializable subdomains.

1.5. Persistent classes must perform hashCode () and equals () methods.

1.6. Applications running on the server should not repeat the capabilities provided by the application server. For example: Creating a thread in the J2E framework

1.7. Classes with a pointer must perform their own copy methods.

1.8. All non-static variables must be given an initial value.

1.9. In an instance, self-deletion should be avoided.

1.10. Type casting should only be carried out for compatible types.

1.11. Data should not be shortened while transporting data in the memory, the data to be moved to the two ends (source and target) should be guaranteed to be compatible in terms of type, size and capacity.

1.12. There should be no function with an indefinite number of parameters. The input parameters received must be used with data structures and for each parameter should have a specific data structures. For example, if a function that should take 3 parameters, such as id, name and description, should not take all parameters as a single parameter separated by special characters.

1.13. The return value must be tested in all resource allocation statements. For example; get memory, get thread, get db connection, and open file etc.

1.14. It should not be tested whether the floating-point numbers are equal.

1.15. Any function should leave back any resources it allocates (in order not to encounter the Memory Leak Problem.)

1.16. The references given to the buffers should be guaranteed to fit in the dimensions reserved for the buffer.

1.17. All data access must be via a central data manager (transaction manager).

1.18. If a singleton pattern is applied in multi-thread environments, the locks must be set up without creating singleton classes.

1.19. Cyclic loop calls should be avoided. For example: If A calls B, B shouldn't call A.

1.20. Superclass should not be aware of subclasses and should not use sub-classes. (Superclass should not call/use sub-class method, its attributes or sub-class name.)

1.21. In special destructor-writeable software languages, classes with virtual methods must have a virtual destructor

1.22. In special destructor-writeable software languages, master classes must have a virtual destructor

1.23. In special destructor-writeable software languages, subclasses must implement virtual destructors in their master class

1.24. The cyclomatic complexity of all modules should be acceptable

1.25. Network resources (IP Address, hostname, port, URL, etc.), user codes, passwords in code shouldn't be hardcoded. Instead, it must be implemented with encryption algorithm and parametric use.

1.26. Logs should not include password or special details of a customer.

1.27. Modules that use resources should also have statements that clear these blocks. This means that pieces of code that specify how to clean resources during garbage collection should also be written.

1.28. There must be a timeout for blocking synchronous calls.

1.29. The access information and passwords of the module / display / sub-programs in the application should not be left as default assigned values.

1.30. Information written in the log should contain information summarizing the situation rather than the general expression.

2. Performance Qualification Requirements:

Under the given conditions, the performance level of a software and the amount of resources that use it are evaluated.

2.1. Client requests must be centralized to reduce network traffic.

2.2. SQL queries that perform sequential searches should be avoided.

2.3. Very large tables on the complex queries should not be used. For example: many tables should be joined with each other, too many subqueries should be avoided.

2.4. Usage of excessive index or multiple indexes in a table should be avoided.

2.5. If large tables and indexes are available in the database, partition should be used.

2.6. If memory is limited, the correct parser should be selected. For example, avoiding the use of the DOM and something like SAX should be used.

2.7. Operations that adversely affect performance (OPEN / CLOSE, object creation, CREATE, object reset, database connection, remote command call, SQL query, etc.) should be avoided in the loop.

2.8. Lazy object creation should be done rather than creating a full object in static blocks.

2.9. It is recommended that the number of SQLs in the middle layer should not be more than 2.

2.10. If the SQL number is more than 2, the store procedure must be used.

2.11. Objects that cannot be changed can also be avoided. For example, it should not be attempted to create a new string from two strings. In such a case, StringBuffer / StringBuilder should be used.

2.12. StringBuilder is recommended because it will be faster rather than StringBuffer if a synchronized structure is not required.

2.13. The reference of unused objects should not be kept. In this case, the object will remain unclean during garbage collection (garbage collection, cleaning of unused objects).

2.14. Keeping heavy objects (requiring large amounts of memory) in the session should be avoided.

2.15. The use of static variables and static objects should be avoided. If it is necessary to use, it should be used as multi-tread environment.

2.16. Instead of using a static connection, the required connection must be taken from the connection pool.

2.17. It is necessary to specify the names of the fields to be queried at the point of querying from the database and query only needed data.

2.18. When bulk data is retreival required from the database, the paging structure must be used. It is necessary to limit the size of the data to query by the interval.

2.19. A call on the provided webservice or screens is expected to return a response within 5 seconds (except for the delays caused by the network devices being accessed).

2.20. For queries written for database operations, prepared statement should be used.

3. Maintenance Feasibility Requirements:

Expresses the effectiveness and ability of making the desired changes in the application, software or system.

3.1. Functions / methods that perform data exchange should only be changed with adjacent layer functions / methods. It should not skip the adjacent layer and make changes with the other layer.

3.2. Too many horizontal layers should be avoided.

3.3. Long code segments should be collected in one place. Copy-paste should prevent the distribution of these pieces of code (Code duplication issue).

3.4. A class's inheritance depth should be limited.

3.5. The number of classes derived from a class must be limited.

3.6. Multiple inheritance should be avoided. For example, a class should not be derived from both A, B, and C.

3.7. The data update / insertion features should be able to be stored, restricted, and encapsulated.

3.8. Data members of classes should not be public.

3.9. A class's use of other classes (fan-out) should be restricted. The threshold value should be ≤ 5.

3.10. Cyclic loop calls should be avoided. For example, If A calls B, B should not call A.

3.11. Instead of multiplying the same code snippet by typing at different points, it should be written to a single point and use from necessary points as reference.

3.12. Instructions should not be closed as comments.

3.13. Files should not contain over 1000 lines of code.

3.14. Indices (counter, index) should not be changed within the loop.

3.15. GO TO, CONTINUE and BREAK should not be used except the switch cycle.

3.16. Cyclomatic complexity should be limited. It is recommended not to exceed the complexity value of 12.

3.17. Depending on the number of database / file operations, complexity should be checked.

3.18. The number of parameters passed by a function / method must be less than 7.

3.19. Other than trivial idioms (literal) should not be hard code.

3.20. All error messages should be kept in a central location. There must be no development, deployment, or system stop to change an error message.

3.21. Line length should not be more than 80 lines.

3.22. For value assignments, d = (a = b + c) + r; substitute a = b + c; d = a + r; writing should be applied.

3.23. Use of parentheses: if (a == b && c == d) instead of if ((a == b) && (c == d)).

3.24. There should not be more than one return statement in the method / functions.

3.25. There should be no duplicated code fragments.

3.26. All software developers must use the same code editor (code formatter / beautifier).

3.27. The code must be passed through the code analysis tools and the code formatter before it is sent to the repository. The code must be sent after the errors of the analysis are cleared.

3.28. The number of lines of code in Method / Function / Procedures should not be higher.

3.29. Each class / function / method / procedure should have a brief description of what the purpose of the piece of code serves.

3.30. Based on the development, all necessary information about how to make any changes or management by the user or admin during the life cycle including the initial installation of the application / development should be shared with the package delivery.

Annex II: Yazılım Kod Kalitesi Gereksinimleri

YAZILIM KOD KALİTESİ GEREKSİNİMLERİ

Yazılım, sistem veya bileşenlerinin, belirli bir ortamda, belirli bir zaman dilimi içinde kendilerinden beklenilen işlevleri yerine getirebilme yeteneği olarak tanımlanmaktadır. Bu doküman şirket içi veya ısmarlama şekilde geliştirilen yazılımlarda kaliteyi sağlamak amacıyla hazırlanmıştır. Aşağıda kalitenin sağlanması için gereksinimler sıralanmıştır.

Kod kalitesine ait gereksinimler 4 başlık altında toplanmıştır.

1. Güvenilirlik
2. Performans Yeterliliği
3. Bakım Yapılabilirlik
4. Güvenlik

1. Güvenilirlik Gereksinimleri:

1.1. Try, Catch, Finally, exception gibi veya buna benzer hata yakalama blokları boş olmamalıdır.

1.2. Catch blokları ile yakalanan hatalar hiçbir işlem yapılmadan rethrow edilmemeli. Yakalanan hatalar loglanmalıdır.

1.3. Veritabanında çalıştırılacak Insert, Update, Delete, Create Table veya Select komutlarını içeren fonksiyonlar ve procedürler hata yönetimi (exception handling) yapmalıdır.

1.4. Serializable arayüzü gerçekleştiren sınıflar aynı zamanda serializable metodunu sınıfın kendi ve tüm serializable alt alanları için gerçekleştirmelidir.

1.5. Persistent sınıflar hashCode() ve equals() metodlarını gerçekleştirmelidir.

1.6. Sunucu üzerinde çalışan uygulamalar uygulama sunucusu tarafından sağlanan yetenekleri tekrarlamamalıdır. Örneğin: J2E çatısı içinde thread oluşturmak

1.7. Pointer içeren sınıflar kendi copy metodlarını gerçekleştirmelidirler.

1.8. Tüm non-static değişkenler için bir başlangıç değeri verilmelidir.

1.9. Bir instance içinde kendini silme işleminden kaçınılmalıdır.

1.10. Tip çevrimleri sadece uyumlu tipler için yapılmalıdır.

1.11. Hafıza içerisinde data taşınırken kısaltılmamalı, datanın taşınacağı iki ucun (kaynak ve hedef) uyumlu boyutlarda olması garanti altına alınmalıdır.

1.12. Belirsiz sayıda parametreye sahip herhangi bir fonksiyon olmamalıdır. Alınan giriş parametreleri veri yapıları üzerinden ve her bir parametreye özel veri yapıları kullanılmalı. Örneğin, id, name ve description gibi 3 parametre alması gereken bir fonksiyon tüm parametreleri tek bir string içinde özel karakterlerle ayrıştırılmış şekilde tek bir parametre olarak almamalıdır.

1.13. Bütün resource allocation ifadelerinde geri dönen değer test edilmeli. Bu ifadeler get memory, get thread, get db connection ve open file ifadelerini içermektedir

1.14. Floating point sayılarının eşit olup olmadığı test edilmemelidir.

1.15. Herhangi bir fonksiyon ayırdığı her kaynagı geri bırakmalıdır.(Memory Leak Problemi ile karşılaşmamak için.)

1.16. Bufferlara verilen referansların buffer için ayrılan boyutlara sığması garanti altına alınmalıdır.

1.17. Bütün data erişimi, merkezi bir data manager (transaction manager) üzerinden olmalıdır.

1.18. Multi-thread ortamlarda singleton pattern uygulanıyorsa, locklar singleton sınıflar oluşturulmadan kurulmalı.

1.19. Dairesel çağrılardan kaçınılmalıdır. Örneğin: A B yi çağırırken B de A yı çağırmamalıdır.

1.20. Superclass lar subclass lardan haberdar olmamalı ve sub-class ları kullanmamalı. (Superclass sub-class ın methodunu çağırmamalı, attributelerini kullanmamalı ve sub-class adını kullanmamalı.)

1.21. Özel destructor yazılabilen yazılım dillerinde, sanal method içeren sınıflar sanal destructorada sahip olmalıdır

1.22. Özel destructor yazılabilen yazılım dillerinde, ana sınıflar sanal destructora sahip olmalıdır

1.23. Özel destructor yazılabilen yazılım dillerinde, alt sınıflar ana sınıflarındaki sanal destructorları implement etmelidir

1.24. Bütün modüllerin cyclomatic complexity leri kabullenilebilir seviyede olmalı

1.25. Network kaynaklarının (IP Adresi, hostname, port, URL vs), kullanıcı kodları, şifreleri kod içerisinde ve açıkça yazılarak (hard code) kullanılmamalıdır. Bunun yerine encryption algoritmasından geçirilmeli ve parametrik kullanım yapılmalıdır.

1.26. Loglara şifre bilgisi veya müşteri özelindeki özel bilgileri yazılmamalıdır.

1.27. Kaynakların kullanıldığı modüllerin ayrıca bu blokları temizleyen ifadelerinin de olması gerekmektedir. Yani garbage collection sırasında kaynakların nasıl temizleneceğini belirten kod parçaları da yazılmalıdır.

1.28. Senkron çağrıları engellemede bununla ilgili zamanaşımı olmalıdır.

1.29. Uygulama veya çözüm içerisindeki modül/ekran/alt-programların erişim bilgileri ve şifreleri default atanan değerler olarak bırakılmamalıdır.

1.30. Loglara yazılan bilgilerde genel ifadeden çok, o durumu özetleyen açıklama ve bilgiler yer almalıdır.

2. Performans Yeterliliği Gereksinimleri:

Verilen şartlar altında, bir yazılımın performans seviyesi ve kullandığı kaynak miktarını etkileyen özelliklerin değerlendirilmesidir.

2.1. Ağ trafiğini düşürmek için client istekleri merkezileştirilmelidir.

2.2. Sıralı arama yapan SQL sorgularından kaçınılmalıdır.

2.3. Çok büyük tablolar üzerinde kompleks sorgulamalar yapılmamalı. Örneğin: birçok tablonun birbiri ile join yapılması, çok fazla alt sorgu olması, joinlerde performansı düşüren sıralamalar yapılmasından kaçınılmalıdır.

2.4. Aşırı büyük index veya bir tabloda birden fazla index kullanımından kaçınılmalıdır.

2.5. Veritabanında buyük tablolar ve indexler uygun ise partition yapılmalıdır.

2.6. Hafızanın kısıtlı olması durumunda doğru parser seçimi yapılmalıdır. Örneğin DOM kullanmaktan kaçınıp SAX gibi bir şey kullanılmalıdır.

2.7. Performansı olumsuz etkileyen operasyonlar (OPEN/CLOSE, nesne yaratımı, CREATE, obje sıfırlaması, database bağlantı yapılması, uzaktan komut çağrısı, SQL sorgusu vs) döngü (loop) içinde kullanmaktan kaçınılmalıdır.

2.8. Statik bloklar içinde full nesne oluşturulmasından ziyade lazy nesne oluşturması yapılmalıdır.

2.9. Orta katmandaki SQL'lerin sayısı 2'den fazla olmaması tavsiye edilmektedir.

2.10. SQL sayısı 2'den fazla ise, store procedure kullanılmalıdır.

2.11. Değiştirilemeyen nesnelerden, ayrıca nesne yaratılmasından kaçınılmalı. Örneğin, javada iki string'den yeni bir string oluşturmaya çalışılmamalıdır. Böyle bir durumda StringBuffer/StringBuilder kullanılmalıdır.

2.12. Synchronized bir yapı gerekmiyorsa StringBuffer yerine daha hızlı olacağı için StringBuilder tavsiye edilmektedir.

2.13. Kullanılmayan nesnelerin referansı tutulmamalıdır. Bu durumda o nesne garbage collection (çöp toplama, kullanılmayan nesnelerin temizlenmesi) sırasında temizlenmeden kalacaktır.

2.14. Ağır nesnelerin (büyük miktarda memory gerektiren) session'da tutulmasından kaçınılmalıdır.

2.15. Statik değişken ve statik nesnelerin kullanılmasından kaçınılmalıdır. Eğer kullanılacaksa bu singleton olarak multi-tread ortamında kullanılmalıdır.

2.16. Statik bağlantı kullanmaktansa, ihtiyaç duyulan bağlantı, bağlantı havuzundan (connection pool) alınmalıdır.

2.17. Veritabanından veri çekilmesi noktasında çekilecek alanların isimleri belirtilerek ve ihtiyaca yönelik verilerin çekilmesi gerekmektedir.

2.18. Veritabanından toplu veri çekilirken, sayfalama yapısı kullanılmalıdır. Aralık verilerek çekilecek olan veri boyutunun sınırlandırılması gerekmektedir.

2.19. Sunulan webservis veya ekranlardaki bir çağrının 5 saniyeden önce cevap dönmesi beklenmektedir.(Gidilen uç sistemler, erişim sağlanan network cihazlarından kaynaklanan gecikmeler hariç)

2.20. Kod içerisinde veritabanı işlemleri için yazılan sorgularda prepared statement kullanılmalıdır.

3. Bakım Yapılabilirlik Gereksinimleri:

Uygulamada, yazılımda ya da sistemde yapılmak istenilen değişikliklerin yapılabilme etkinliğini ve yeteneğini ifade eder.

3.1. Data değişimi yapan fonksiyonlar sadece bitişik katman fonksiyonları ile değişim yapmalıdır. Bitişik katmanı atlayıp diğer katmanla değişim yapmamalıdır.

3.2. Çok fazla yatay katmanlardan kaçınılmalıdır.

3.3. Uzun kod segmentlerinin bir yerde toplanması sağlanmalıdır. Copy-paste ile bu kod parçalarının dağıtılması engellenmelidir.(Code duplication sorunu)

3.4. Bir sınıfın kalıtım derinliğini sınırlanmalıdır.

3.5. Bir sınıftan türeyen sınıf sayısı sınırlanmalıdır.

3.6. Çoklu kalıtımdan sakınılmalıdır. Örneğin bir class hem A, hem B, hem de C den türememelidir.

3.7. Verinin güncellenmesi/eklenmesi özellikleri diğerlerinden saklanabilmeli, erişim sınırlandırılabilmeli, encapsule olmalıdır.

3.8. Sınıfların veri üyeleri public olmamalıdır.

3.9. Bir sınıfın başka sınıfları kullanma (fan-out) değeri kısıtlanmalıdır. Eşik değeri ≤ 5 olmalıdır.

3.10. Dairesel çağrılardan kaçınılmalıdır. Örneğin: A B yi çağırırken B de A yı çağırmamalıdır.

3.11. Aynı kod parçacığını farklı noktalara yazarak çoklamak yerine tek bir noktaya yazılmalı ve diğer noktalar referansını kullanmalıdır.

3.12. Komutlar (instructions) yorum olarak kapatılmamalıdır.

3.13. Dosyalar 1000 satırın üzerinde kod içermemelidir.

3.14. İndisler (sayaç, index) döngü içerisinde değiştirilmemelidir.

3.15. Switch döngüsü dışında GO TO, CONTINUE ve BREAK kullanılmamalıdır.

3.16. Cyclomatic karmaşıklık limitlenmelidir. Karmaşıklık değeri 12'yi geçmemesi tavsiye edilmektedir.

3.17. Veritabanı/dosya işlemlerinin sayısına bağlı olarak karmaşıklık kontrol edilmelidir.

3.18. Bir fonksiyon tarafından geçirilen parametre sayısı 7'den az olmalıdır.

3.19. Önemsiz kalıp deyimler (literal) haricindekiler hard code edilmemelidir.

3.20. Tüm hata mesajları merkezi bir yerde tutulmalıdır. Bir hata mesajının değiştirilmesi için geliştirme, deployment ya da sistem kesintisi gerekmemelidir.

3.21. Satır uzunluğu(line length) 80'den fazla olmamalıdır.

3.22. Değer atamaları için d = (a = b + c) + r; yerine a = b + c; d = a + r; yazım şekli uygulanmalıdır.

3.23. Parantez kullanımı: if (a == b && c == d) yerine if ((a == b) && (c == d)) şeklinde olmalıdır.

3.24. Method/fonksiyonlarda birden fazla return satırı bulunmamalıdır.

3.25. Duplike kod parçaları bulundurulmamalıdır.

3.26. Yazılım geliştiricilerin tamamı aynı kod düzenleyici (code formatter/ beautifier) kullanmalıdır.

3.27. Kod repository e gönderilmeden önce kod analiz araçlarından ve code formatter dan geçirilmelidir. Analiz sonucu çıkan hatalar giderildikten sonra kod gönderilmelidir.

3.28. Method/Fonksiyon/Procedure lerdeki kod satır sayısı yüksek olmamalıdır.100-120 satırdan daha fazla kod içermemelidir.

3.29. Her class/fonksiyon/method/procedure ile ilgili kod parçasının ne amaca hizmet ettiği yorum alanında kısaca anlatılmadır.

3.30. Yapılan geliştirmeye istinaden uygulamanın/geliştirmenin ilk kurulumu dahil olmak üzere hayat döngüsü boyunca kullanıcısı veya admin tarafından yapılabilecek her türlü değişikliğin ve yönetiminin nasıl yapılacağı ile ilgili gerekli tüm bilgiler paket teslimi ile birlikte doküman olarak paylaşılmalıdır.

RESUME

Name Surname : Mehmet Yıldız

Place of Birth : Halfeti/Şanlı Urfa

Date of Birth : 1986

Education : Doğuş University, Computer Eng.(%100 scholarship), Feb.2010

Experiences :

2015-… Solution Development Senior Specialist, Türk Telekom A.Ş

2012-2015 Senior Software Engineer, NETAŞ Telecommunication

2010-2012 Software Engineer, Huawei Telecommunication